High Performance Computing as a Combination of Machines and Methods and Programming

Claude Tadonki

UNIVERSITÉ PARIS SUD | FACULTÉ DES SCIENCES D'ORSAY

MINES ParisTech

Année 2013

HABILITATION A DIRIGER DES RECHERCHES

Préparée à

la Faculté des Sciences de l'Université Paris-Sud d'Orsay

et à

l'Ecole des Mines de Paris – Centre de Recherche en Informatique

Spécialité : **Informatique Scientifique**

High Performance Computing as a Combination of Machines and Methods and Programming

Claude TADONKI

Soutenue à l'Université Paris-Sud le 16 mai 2013 devant le jury :

Abdel LISSER, Professeur, Paris XI / LRI (France), Président du jury
Christine EISENBEIS, Directeur de Recherche, INRIA-LRI (France), Examinateur
Nelson MACULAN, Professeur, Université Fédérale de Rio de Janeiro (Brésil), Rapporteur
Jean-Philippe VIAL, Professeur Honoraire, Université de Genève - Ordecsys (Suisse), Examinateur
Jean-Luc GAUDIOT, Professeur, Université de Californie (Etats-Unis), Rapporteur
Luc GIRAUD, Directeur de Recherche, INRIA-CERFACS (France), Rapporteur
Gilbert GROSDIDIER, Directeur de Recherche, *LAL* / IN2P3 / CNRS (France), Examinateur
Olivier PENE, Directeur de Recherche, LPT d'Orsay – CNRS (France), Examinateur

0.1 Summary

High Performance Computing (HPC) aims at providing reasonably fast computing solutions to both scientific and real life technical problems. Many efforts have indeed been made on the way to powerful supercomputers, both generic and customized configurations. However, whatever their current and future breathtaking capabilities, supercomputers work by brute force and deterministic steps, while human mind works by few strokes of brilliance. Thus, in order to take a significant advantage of hardware advances, we need powerful methods to solve problems together with highly skillful programming efforts and relevant frameworks.

The advent of multicore architectures is noteworthy in the HPC history, because it has brought the underlying concept of multiprocessing into common consideration and has changed the landscape of standard computing. At a larger scale, there is a keen desire to build or host frontline supercomputers. The yearly Top500 ranking nicely illustrates and orchestrates this supercomputers saga. For many years, computers have been falling in price while gaining processing power often strengthened by specialized accelerator units. We clearly see that what commonly springs up in mind when it comes to HPC is computer capability. However, this availability of increasingly fast computers has changed the rule of scientific discovery and has motivated the consideration of challenging applications. Thus, we are routinely at the door of large-scale problems, and most of time, the speed of calculation by itself is no longer sufficient. Indeed, the real concern of HPC users is the *time-to-output*. Thus, we need to study each important aspect in the critical path between inputs and outputs, and keep striving to reach the expected level of performance. This is the main concern of the viewpoints and the achievements reported in this book.

The document is organized into five chapters articulated around our main contributions. The first chapter depicts the landscape of supercomputers, comments the need for tremendous processing speed, and analyze the main trends in supercomputing. The second chapter deals with solving large-scale combinatorial problems through a mixture of continuous and discrete optimization methods, we describe the main generic approaches and present an important framework on which we have been working so far. The third chapter is devoted to the topic accelerated computing, we discuss the motivations and the issues, and we describe three case studies from our contributions. In chapter four, we address the topic of energy minimization in a formal way and present our method based on a mathematical programming approach. Chapter five debates on hybrid supercomputing, we discuss technical issues with hierarchical shared memories and illustrate hybrid coding through a large-scale linear algebra implementation on a supercomputer.

0.2 Preface

The current dissertation aims at providing a consistent and chronological view of my research background and corresponding achievements, starting with a panoramic view of the supercomputers landscape and surrounding activities.

At the earlier stage of my higher education, I was trained in pure mathematics and more generally in fundamental sciences. Starting in that way has certainly influenced my taste for formal approaches, and forged my ability to understand and move deeper into mathematically oriented topics. Then, I get introduced into computer sciences, focusing on *algorithm, complexity, scientific programming*, and *parallel computing*. The outcome of this step towards advanced scientific research is a PhD in computer science that I got in march 2001. The title of my PhD dissertation was "Contributions to Parallel Computing", where I presented my results in *parallel linear algebra*, *systolic computation*, and *methodology for parallel scheduling*. I was actively involved in three distinct teams working in *numerical computation*, *discrete mathematics*, and *integrated parallel architectures* respectively. This was a great chance to strengthen my scientific culture and to have a broad range of technical contributions. This was also the occasion to see how different aspects of computer science could be connected in order to achieve more efficient solutions or more robust methodology. This is mainly the hallmark of my scientific route.

After my PhD, I moved to the university of Geneva (Switzerland) for a postdoctoral position. The research topics of the host laboratory (LOGILAB, headed by Pr Jean-Philippe Vial and Pr. Alain Haurie), included *mathematical programming*, *non-differentiable optimization*, and *operation research*. It was expected of me to study the linear algebra kernel of the *cutting planes method* and help implementing them as efficient as possible at the level of high performance computing state-of-the-art. Another (but indirect) expectation was to benefit from my background in *combinatorial optimization* to improve the heuristics that will be used to solve subproblems. This was really a very exciting and fruitful adventure. Indeed, from a personal point of view, the area of *continuous optimization*, by itself and though its connection with combinatorial optimization, was a nice complementary skill that would allow me to have a more mature capability to tackle large-scale combinatorial problems. In addition, I attended several national and international scientific meetings, where I could meet notorious scientists in the field of *optimization* and *operation research*. The main outcome of this postdoctoral step was the design of a flexible *oracle based* solver that is used to solve non-differentiable optimization problems. For combinatorial optimization problems, the solver can be used to solve the linear relaxation at each nodes of the *branch-and-bound* or one of its variants (branch-and-cut, branch-and-price, $\cdots$). Other contributions include the application of the method to solve number of operation research problems, and matrix computation improvements related to the kernel of the solver. The second chapter of the document is devoted to this part of my background and potential perspectives.

Next to my stay at the LOGILAB (around 3 years), I was hired at *Centre Universitaire Informatique* of the university of Geneva, precisely at *laboratory of theoretical computer science* (TCS-Lab, headed by Pr. Rolim Jose). The laboratory was involved in cutting-edge research in the *foundations of computation* and in *parallel distributed computing*. My main contributions during my stay the TCS-Lab were on a formal study of the energy minimization problem (modeling and power-aware scheduling). I was able to use my recent skill in mathematical programming to develop a mixed integer programming model for power consumption of computer programs. This contribution is presented in chapter 4, where we discuss about

the topic of power-aware computing. We again used the mathematical programming approach to solve the dual-power management in sensors networks. We clearly see how rewarding was my investment in the field of mathematical programming. We also developed efficient distributed algorithms for sensors networks and studied the localization problem. My activities at the TCS-Lab, through international projects, gave me the opportunity to cooperate with number of reputed laboratories and talented scientists. In addition, I could attend several international scientific events as a speaker. Moreover, I was able to initiate funded scientific projects in the field of *power aware computing*.

After my two years at the TCS-Lab, I joined the European Laboratory of Molecular Biology at Grenoble (France), in the team of Raymond Ravelli and Florent Cipriani, where I was concerned with mathematical modeling and computational engineering to study the effect of radiation damage in X-ray synchrotron crystallography. The goal was to provide an analytical model for the radiation damage and then find a way to refine collected data by means of computer processing. This was an opportunity for me to work together with people from other disciplines related to structural genomics, and get familiar with experimental research and distributed high-throughput computing.

Next, I moved to the Institute of Fundamental Electronics at University of Paris-Sud Orsay (France), working with Lionel Lacassagne on automatic code optimization and deployment on various parallel architectures. Our aim was to understand, trough an intensive benchmark, the key point in the performance of parallel multi-level memory machines. Based on a unified model of major applications classes, and a model of the target architecture, we studied systematic ways to structure the parallel program in order to reach optimal performances. This was a kind of comeback into heart of *parallel computing*. Indeed, two years later, still within the university of Orsay, I joined the Laboratoire Accélérateur Linéaire (LAL), whose the main activity is on cutting edge research in particles physics, nuclear physics, and astrophysics. At the LAL, in collaboration with pluridisciplinary team, we were involved with HPC investigations related to LQCD (Lattice Quantum ChromoDynamics) simulations at the highest scale. Chapter 3 reports my contributions in the fields of *accelerated computing*, which is an approach I suggested for local LQCD calculations and also for heavy image processing applications. Our efforts on large-scale LQCD simulations (with Gilbert Grosdidier, Christine Eisenbeis, Olivier Pène, Denis Barthou,$\cdots$), involved a broad range of complementary HPC topics (*parallel algorithm, SIMD, ill-conditioned matrix computation, supercomputing, high-throughput computing, failure, accelerators*).

I currently hold a research position at the Centre de Recherche Informatique (CRI, headed by François Irigoin) of the Ecoles des Mines de Paris (France) since 2011. My main research topics include *High Performance Computing, Operation Research, Matrix Computation, Combinatorial Algorithm and Complexity, Scientific and Technical Programming, Automatic Code Transformations*. In addition to my pure research activities, I use to initiate and drive various scientific projects and national/international collaborations. I teach CS courses, mainly at a higher level, in different kinds of institution including industries. In addition, I use to supervise PhD students and be part of PhD boards. I am active member of well established scientific corporations and reviewer of number of international journals and conferences.

Thought this document, I hope to provide the substantial part of my past research achievements, and my personal opinion about the trends in scientific research, and what I found to be potentially interesting research axes.
My Personal web page is located at `www.omegacomputer.com/staff/tadonki`

0.3 Acknowledgment

It is always a difficult exercise when it comes to nominate people we wish to thank for an heterogeneous accomplishment that covers a long period of time. Indeed, many people have been involved in my way to this achievement.

Thanks to those who were directly involved in the final phase of the process. Paris-Sud university who approved my application and thus to host this episode of my career. Christine Eisenbeis (INRIA-Saclay), to have kindly supervised the required actions and more generally for her uncountable willingness to provide me every requested support related to my research and collaboration activities. Dominique Gouyou-Beauchamps, for his efforts to validate each step following the rule of Paris-Sud university. Administratives, Stephanie Druetta and Sylvie Jacquemart, for their administrative assistance. The Mines Paristech institute, through François Irigoin (head of the Mines Paristech/CRI), for its financial support and the opportunity that led to my current position. The referees, Nelson Maculan (University of Rio de Janeiro), Jean-Luc Gaudiot (University of California) and Luc Giraud (INRIA-CERFACS), for their detailed and instructive reviews provided in due time. Abdel Lisser (PARIS XI), who accepted to act as president of my defense jury.

Working with Gilbert Grosdidier (LAL) and Olivier Pène (LPT) on the ANR project PETAQCD was very rewarding, with exciting large-scale computing challenges and a good multidisciplinary collaboration. I use this occasion to thank the whole PETACQD group (members and institutions), I have appreciated our discussions and regular meetings. Thanks to LAL, CNRS, and IN2P3 for their institutional support.

I wish to thank Lionel Lacassagne (Digiteo) for our collaboration on image processing algorithms and low-level code optimization including SIMD programming, with good team partners Joel Falcou, Tarik Saidani, and Khaled Hamidouche. This occurred within the frame of another ANR project named OCELLE and the System@tic project TERAOPS, right after my stay at the European Laboratory of Molecular Biology for which I thank Raymond Ravelli and the administrative staff.

I am grateful to Jean-Philippe Vial (University of Geneva/Ordecsys) for having initiated my Geneva saga and for what we did on convex optimization. Our collaboration, which included good colleagues Cesar Beltran, Oliver Péton, Oliver du Merle, and Frédéric Babonneau to name a few, led me to modern optimization techniques, thus giving me the right skill to efficiently tackle large-scale combinatorial problems. Thanks to Alain Haurie (University of Geneva/Ordecsys) for the opportunity he gave me to work with him on the same laboratory, in a project including Laurent Drouet, Alain Dubois and Daniel Zachary. My stay at the university of Geneva was extended by an enthusiastic collaboration with Jose Rolim (Computer Science Center), I really appreciate all he did to facilitate my work and my professional initiatives.

Thanks to Eugene Ressler (Westpoint) and Patrice Quinton (ENS Cachan) for their support concerning my application.

Thanks to all my colleagues, collaborators and friends so far. I have appreciated to see most of you being sincerely delighted by this event.

A special and affective acknowledgment to my whole family. Indeed, I never feel alone with such a close attention, let's share this achievement together.

0.4 Special dedication

In memory of Jean-Tadonki (1939-2001)

Contents

Chapter 1

The landscape of supercomputers

1.1 The landscape of high performance computing

High Performance Computing has been on the spotlight over the last decade, driven by users clamor for more powerful systems and more exciting applications [30, 7, 26, 18]. Significant technical changes have occurred, and noteworthy improvements have been done at various levels, thus pushing the limit of both standard computers and supercomputers. This phenomenon has even changed the rules of scientific discovery. Indeed, large-scale computation is now commonly considered in order to assess if a theory is consistent with experimental results, to question a large collection of data, or to understand a given mechanism through high precision simulations [20, 12, 11, 37, 13]. At the processor level, the sequential von Neumann execution model has governed the computing landscape for more than half century. Thus, the answer for more efficient processing was either a more powerful single-thread processor or an aggregation of cooperative machines. Hardware designers have really strived to increase processor capabilities at different levels including clock speed (also referred to as frequency), memory size and bandwidth, mass storage capacity, and power consumption. Regarding parallel computers, they were mainly built by aggregating many standard processors with a specific interconnect, thus expensive and very heavy to maintain. Thereby, and also due to the need of a particular skill, parallel computing, which was so far the unique choice for high performance computing, had a very limited effective consideration, although intensive efforts at the fundamental level. Back to the processor level, chip designers have always strived to stay ahead of Moore's Law, which prescribes that processor transistor count doubles every two years [10]. This was still possible by adding transistors and logic to the standard CPU and increasing clock frequencies, until it becomes exceedingly impractical because of the power wall associated to the increase of processor frequency. Therefore, leading vendors considered multicore processor strategies, thus opening the door to the multicore era.

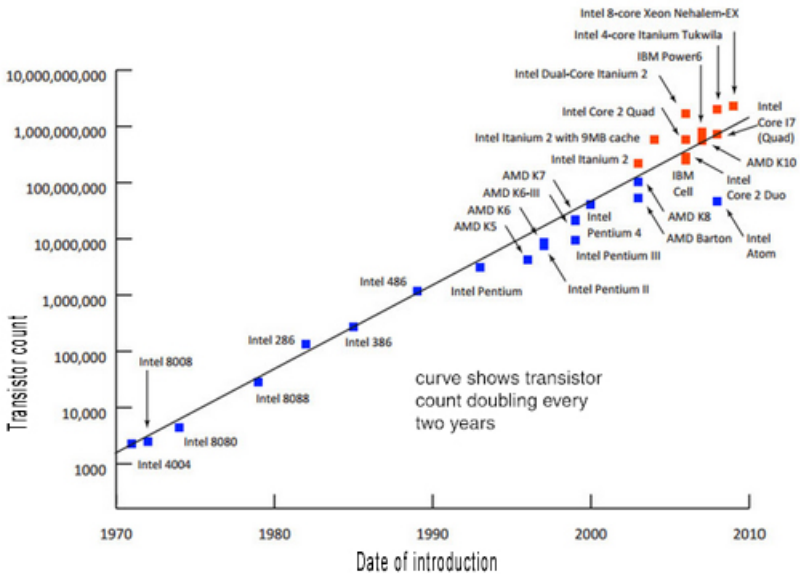


Figure 1.1: Microprocessor Transistor Counts 1970-2010 & Moore's Law

From that inflexion point in the evolution of computer systems, things are changing dramatically, including the emergence of new hardware devices. With the advent of multicore

processors, manufacturers have taken that opportunity to keep providing increasingly powerful processors even to ordinary users, provided that they really transition to parallel computing. Thereby, the notion of parallelism is extending to a wider audience, and will soon or later become a key item in computer science and engineering curricula. Multicore processors are being actively investigated and manufactured by major computer-processors vendors. At present, most contain between 2 to 16 cores, and a few contain as many as 64 to 80 cores (so-called many-core). For the programmer, in addition to requirement of designing multi-threaded codes, now has to face a more complex memory system. Indeed, the memory available on multicore processors has several levels, different packaging and management policies. Figure 1.2 displays and example with the Nehalem architecture.

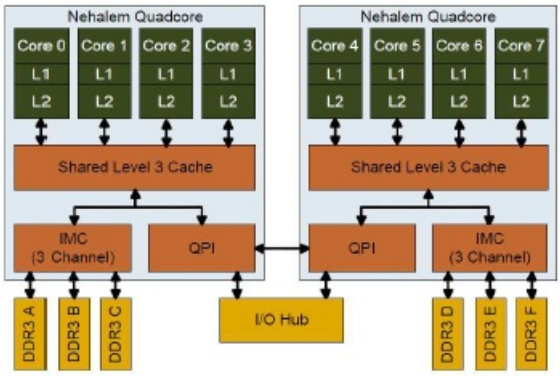


Figure 1.2: Nehalem memory hierarchy

Memory complexity remains a serious challenge both from the hardware and the software standpoints. Indeed, the part due to memory accesses and management in the sustained performance with common applications is quite significant, especially with stencil computation (image processing, simulations based on Cartesian space modeling, discrete iterations, to name a few). In addition to optimizing memory traffic, the programmer now needs to care about cache memories sharing, with a direct consequence on the performance scalability.

In addition to the absolute performance and scalability issues with conventional (multicore) processors, power consumption has quickly become another critical point. The concern is still to compute quite quickly, so as to save energy by reducing the overall running time. The idea that has come in mind to tackle this is the use of accelerators. An accelerator is a specialized unit dedicated to a specific kind of tasks that will be executed with an unbeatable performance. The Graphic Processing Unit (GPU) is one of such devices (see [9] for a survey on the GPU history). As its name implies, the GPU was originally developed for accelerating graphics processing. The demand for increasingly realistic visual content in video games and other applications compelled GPU designers to endow their chips with the ability to apply complex physical equations to the task of producing convincing renderings of real-world phenomena. Eventually, game programmers and GPU manufacturers realized that such achievements for "game physics" could also apply to other fields [38]. The emergence of graphics processing unit (GPUs) as more of a general-purpose computational processor has improved the performance of certain types of computations [32, 29]. Some applications have shown performance improvements ranging from 2x (twice as fast) to over 100x (100 times

faster) [39]. From these numbers, it is pretty obvious why GPUs are so exciting, even if programming them is not intuitive.

Interconnecting a large number of powerful multicore processors (probably accelerated) with a high speed network is leading to impressive supercomputers. The current horizon is the "exascale" [28], which is expected by 2018. Supercomputers are doing ground-breaking work that might not be possible without them, and this has changed the rules of science and industry. With computing possibilities running up against the far edge of current technology, researchers are looking for new ways to shrink processors, combine their power, and gather enough energy to make them all work efficiently. Computational capabilities are nowadays an essential part in cutting-edge scientific and engineering experiments. The capability to analyze and predict from huge amount data has incredibly improved with the use of supercomputers. Neuroscientists can evaluate a large number of parameters in parallel to find good models for brain activity; automobile manufacturers can perform more realistic crash simulation to improve safety; astronomers can analyze different regions of the sky in parallel to search for supernovae; nuclear and particle physics are moving beyond common belief with large-scale simulations; search engines can launch parallel search across large-scale clusters of machines and instantly aggregates the results, thus reducing the latency of each request while improving relevance and accuracy; cryptography and computer systems security will benefit from the computation of gigantic prime numbers; researchers in artificial intelligence are trying to use large supercomputers to replicate (or surpass) a high-functioning human's ability to answer questions; social networking services are increasing their pervasiveness through large-scale graph processing, text processing or data mining.

While keep striving to provide breathtaking faster computers, designers need to contend with power and energy constraints. For decades, computers got faster by increasing their (aggregated) central processor unit. However, high processor frequency means lot of heat. Indeed, The Fujitsu K Computer, for example, has been using US$10 million of electricity per annum to operate. This question of energy is more crucial as computing are being reported to the "Cloud", which is another innovative and affordable way to fulfill the need of high-range computing facilities. Indeed, Cloud computing offers a great alternative on mass storage, software and computing devices [44, 19, 3]. Federating available computing resources, assuming a fast network, is certainly a valuable way to offer a more powerful computing system to the community. Energy, both dissipated and consumed, is also a critical concern, which is subject to active investigation from both the hardware and software standpoints.

From the programming point of view, harvesting hardware advances to rich the level of cutting-edge research expectations is more challenging. Indeed, beside the ambient enthusiasm around the evolution of supercomputers, the way to peak performances is far from straightforward. In addition to algorithmic efforts to express and quantify all levels of parallelism, specific hardware and system considerations have to be taken into account when trying to provide an efficient, robust, and scalable implementation on (heterogeneous) multicore processors. This has brought an unprecedented level of complexity in program design. Adapting a code for a given architecture or optimize it accordingly requires a complex set of program transformations, each designed to satisfy one or more aspects (e.g. registers, cache, instruction pipeline, data exchanges) of the target system. When the program is complex enough, or when the target architecture is a combination of different processing units (hybrid or accelerated computing), devising highly efficient programs becomes seriously hard. This is the price anyone should be aware of, when it comes to current and future states of high performance computing. The evolution of supercomputers performance is well depicted in

the semi-annual top500 ranking. This has triggered an exciting competition among manufacturers and countries for the fastest supercomputer. Leadership in supercomputing is viewed around the world as a symbol of national economic competitiveness and of technical and scientific leadership. Alongside the ranking announcements, top500 reports provide a valuable collection of quantitative information for global statistics and trend analysis. Figure 1.3, for instance, provides a view on the performances evolution (aggregated and extremes) from the beginning of the top500 ranking.

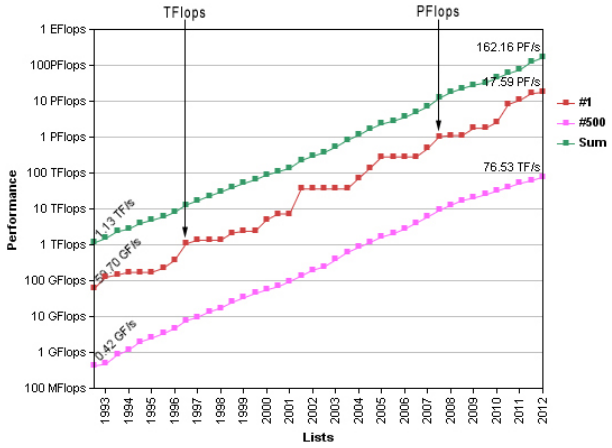


Figure 1.3: Performance evolution overview from the top500

The petaflops barrier was reached for the first time in June 2008 top500 by the IBM *Roadrunner*, nearly ten years after the reach of the teraflops barrier in June 1997 by Intel ASCI Red. The IBM press release used a few analogies to describe the power of Roadrunner, such as "The combined computing power of 100,000 of today's fastest laptop computers"; and, "It would take the entire population of the earth, - about six billion - each of us working a handheld calculator at the rate of one second per calculation, more than 46 years to do what Roadrunner can do in one day." By plain extrapolation, a sustained exascale performance is expected from 2018. It is amazing to see that Titan - Cray XK7, the current world fastest supercomputer, is 294,639 times faster than the top ranked machine of the 1993 top500 edition, the Thinking Machines CM-5/1024. Figure 1.4 is a snapshot of the November 2012 top500 listing, focused on the top ten machines.

TOP10 November 2012

1	Titan - Cray XK7 , Opteron 6274 16C 2.200GHz, Cray Gemini interconnect, NVIDIA K20x	6	SuperMUC - iDataPlex DX360M4, Xeon E5-2680 8C 2.70GHz, Infiniband FDR
2	Sequoia - BlueGene/Q, Power BQC 16C 1.60 GHz, Custom	7	Stampede - PowerEdge C8220, Xeon E5-2680 8C 2.700GHz, Infiniband FDR, Intel Xeon Phi
3	K computer, SPARC64 VIIIfx 2.0GHz, Tofu interconnect	8	Tianhe-1A - NUDT YH MPP, Xeon X5670 6C 2.93 GHz, NVIDIA 2050
4	Mira - BlueGene/Q, Power BQC 16C 1.60GHz, Custom	9	Fermi - BlueGene/Q, Power BQC 16C 1.60GHz, Custom
5	JUQUEEN - BlueGene/Q, Power BQC 16C 1.600GHz, Custom Interconnect	10	DARPA Trial Subset - Power 775, POWER7 8C 3.836GHz, Custom Interconnect

Figure 1.4: Top ten machines of the November 2012 top500

Titan-XK7 is a hybrid supercomputer, means made up by a combination of commodity processors with coprocessors or graphics processing units (GPUs) to form a heterogeneous high-performance computing system. Roadrunner was the world's first hybrid supercomputer, made up with 6,562 dual-core AMD Opteron chips as well as 12,240 Cell chips (on IBM Model QS22 blade servers). Accelerated computing is prevailing over the use of conventional CPU-based architectures, and is certainly the way to power aware supercomputing. Indeed, as supercomputers are to move beyond the petascale and into the exascale, energy efficiency is becoming a major concern. Note that power consumption, as a metric, was not even mentioned in earlier top500 editions. Now, this aspect has come to the spotlight, and there is a so-called *Green500* project, which aims at providing a ranking of the most energy-efficient supercomputers in the world.

In order to figure out what can be expected from a given supercomputer and appreciate its potential, we provide some basic notions.

1.2 Basic quantitative background

1.2.1 Calculating the overall peak performance

The first thing that comes in mind with a supercomputer is its potential performance, also known (and refers to) as *theoretical peak performance*. This is rough calculation of the overall computing power that the considered computer can offer. The items that are mainly considered are

- ◇ The total number of cores (regardless of the packaging)
- ◇ The processor clock rate
- ◇ The length of vector registers (assuming floating point calculations)
- ◇ The possibility (or not) of a one cycle multiply-add (thus, 2 FP calculations per cycle)

Let consider the case of the **IBM-Sequoia** supercomputer that will be fully described later. We have

- ◇ Total number of cores = 1,572,864

- ◇ Processor-core clock rate = 1.6 Ghz
- ◇ Each core has Quad FPU (4-wide double precision vector registers)
- ◇ One cycle multiply-add feature available

This gives

$$1,572,864 \times (1.6 \times 10^6) \times 4 \times 2 = 20.132659 \times 10^{15} \approx 20.14\text{PFlops} \tag{1.1}$$

We point out the fact that memory accesses and interprocessor communication are not counted. The reader should kept it in mind, and be aware that this is where mainly comes the gap between peak and sustained performances. However, the impact of data moves can be attenuated by an overlap with computations, at the price of skillful programming efforts.

1.2.2 Evaluating interprocessor communication

A supercomputer is composed of a large number or computing nodes which need to exchange data (inputs or intermediate results) in order to achieve the global assigned task. As said above, this time cost for interprocessor communication is roughly seen as an additional time over the pure computation time. For a single data communication, there a setup latency and a transmission time, which gives an estimation of the form

$$T_c(L) = \beta + \alpha \times L \tag{1.2}$$

As multiple transfers can occur at the same time, the inverse of the latency (i.e. $1/\beta$) is sometimes referred in the literature as the number of MPI communication that can be launched within a second. The physical network topology and the current data traffic will determine the effective cost [4, 25, 24]. There are more sophisticated cost models in the literature, but they are rarely considered in practice, probably because they are too difficult to handle and the added-value is marginal.

1.2.3 Energy

This a very important measure when it come to supercomputers [3, 4]. Indeed, processing with a supercomputer implies an large aggregation of heavy CPU activities, thus a risk of overheating. Based on the Ohm's Law [1], we have that the *dissipated power* is approximately proportional to the square of the CPU voltage and the CPU frequency, which gives

$$P = CV^2 f, \tag{1.3}$$

where C is *capacitance*, V is *voltage*, and f is *frequency* [43]. It is important to note that those parameters can be changed dynamically at runtime [5], which offers an opportunity for an energy-aware scheduling. The *network* and the *memory* activities also count, but the most important focus is on the pure CPU side. A more detailed and formal analysis of this topic is provided in Chapter 4.

1.2.4 Failure

Failure is a natural fact that arises soon or later with any device. For (super)computers, the main focus is on the CPU failure that is caused by heat dissipation. A nice survey open the topic is presented in [2]. The typical way to contend with this issue is *cooling*. In any case, a prediction of the time to failure is a common measure in the HPC community [41, 42]. The most popular measure is the *mean time between failures* (MTBF). We also have the *mean time to failure* (MTTF). MTBF and MTBF are sometimes used interchangeably, but they are slightly different in the sense MTBF refers to a failure that can be (and will be) repaired, while the MTTF refers to a fatal failure. MTTF also includes the case where the maintenance policy is *replacement* or *removal*, even if the problem can be fixed.

MTBF can be estimated from a time interval by calculating the ratio of the total time measured over the total number of failures observed. This assumes a uniform distribution of failures, but a more sophisticated model could be considered if necessary and possible. For example, if we run a supercomputer with 500 nodes for 600 hours and we find 15 failures, then we have

$$MTBF = \frac{(500 \times 600)}{15} = 20,000 \text{ hours.} \tag{1.4}$$

We now illustrate the state-of-the-art of supercomputers trough a commented description of a selected subset of world-class computing systems, followed by a view on more specific architectures. Our sampling focuses on top-ranked machines, different kinds of architecture (CPU + interconnect), and the packaging (node configuration).

1.3 Selected architectures

1.3.1 TITAN - CRAY XK7

Titan - Cray XK7, a hybrid CPU/GPU supercomputer manufactured by the Cray Company, was ranked world's fastest supercomputer in the November 2012 top500 ranking. The Cray XK7$^{\text{TM}}$, installed at the Department of Energy's Oak Ridge National Laboratory (ORNL / USA), has showed an outstanding 17.59 PFlop/s Linpack performance, that is quadrillions of calculations per second, over a theoretical peak of 27.11 PFlop/s. The machine is made up with 299,008 cores AMD Opteron 6274 (16 cores per node), each core clocked at 2.2 GHz. This aggregation of CPUs is combined with 18,688 NVIDIA Tesla K20 GPUs. The total memory space available is 710 TB, and the total power consumption is around 8.2 megawatts, which yields a remarkable (rank 2) performance/power ratio of 2.14 MFlops/watts. The network is a 3D torus topology based on the efficient *Gemini interconnect*, which is capable of tens of millions of MPI messages per second with 1.5 microsecond latency and a bandwidth of 20 GB/s for point-to-point transmissions.

Figure 1.5: TITAN Supercomputer

Among the set of applications that can notably benefit from the tremendous processing speed of Titan, Oak Ridge National Laboratory reported seismological simulations of the entire Earth (suggested by researchers from Princeton University), direct numerical simulation with complex chemistry to understand turbulent combustion, discrete radiation transport calculation, molecular studies, climate change adaptation and mitigation scenario, to name a few. We think that the presence of GPUs should somehow influence the range of potential applications that can be efficiently ported on such machine. A typically suitable application should allow a coarse grain task partitioning with locally interconnected stream processing nodes.

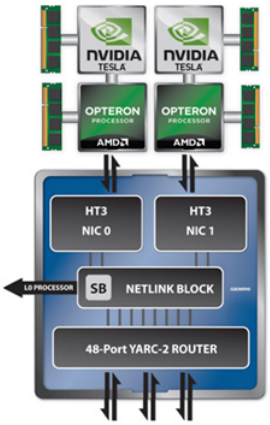


Figure 1.6: GEMINI

1.3.2 IBM SEQUOIA

Sequoia is a world-class IBM BlueGene/Q computer, which was ranked second world's fastest supercomputer in the November 2012 top500 ranking, after being atop in the previous edition. The Sequoia, hosted at the Department of Energy's Lawrence Livermore National Laboratory (LLNL / USA), has showed a distinguished 16.32 PFlop/s Linpack performance (16 thousand trillion calculations per second) over a theoretical peak of 20.14 PFlop/s. The machine, at this time (upgrades are awaited), is made up with 1,572,864 cores (16-cores CPUs), each core clocked at 1.6 GHz, and a total memory of 1573 TB. Another attractive strength of Sequoia is its power consumption, which is estimated at 7.9 megawatts, thus making it a good candidate for high-performance computing and high-throughput computing as well. The network is a 5D torus bidirectional optical network with a bandwidth of 5 GB/s and a latency of 2.5 microseconds.

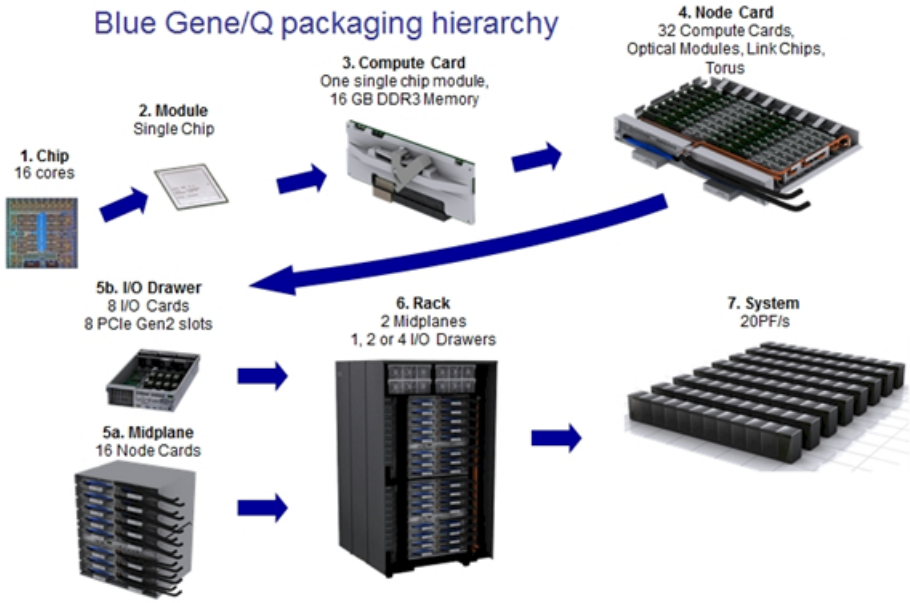


Figure 1.7: Sequoia packaging

The Sequoia is planned to be eventually devoted almost exclusively to simulations aimed at extending the lifespan of nuclear weapons. However, it flexible interconnect makes it a good choice for (block) stencil computation like the *Lattice Quantum ChromoDynamics* (LQCD) or Discrete Partial Differential Equation (DPDE). More classical applications are also considered like semiconductor and silicone design, financial modeling, climate and weather studies. The modest clock rate of each individual core suggests that the machine could be considered for large scale memory bounded applications. Moreover, the noteworthy low power consumption of the BlueGene/Q makes it clearly adapted for high-throughput computation, with an affordable energy and maintenance cost.

1.3.3 Fujitsu K-COMPUTER

K-COMPUTER is a Fujitsu supercomputer, which was ranked third world's fastest supercomputer in the November 2012 top500 ranking, after being atop in the 2011 edition. The K-Computer, hosted at RIKEN Advanced Institute for Computational Science (AICS / Japan), has showed an impressive 10.5 PFlop/s Linpack performance (nearly 11 thousand trillion calculations per second) over a theoretical peak of 11.2 PFlop/s. The heart of the K computer consists of 88,128 SPARC64^{TM} VIIIfx 8-cores CPUs, thus a total of 705,024 cores. The overall global memory sums up to 1410 TB. The power consumption is around 12.7 megawatts, which yields a relatively high power per core compared to other machines of the top ten. However, we think that this controversy power consumption is well compensated by the close gap between sustained and peak performances. The K computer's network, called *Tofu*, uses an innovative structure called "6-dimensional mesh/torus" topology with a total throughput of about 5 GB/s and a microsecond latency for a point-to-point communication between two neighbor nodes.

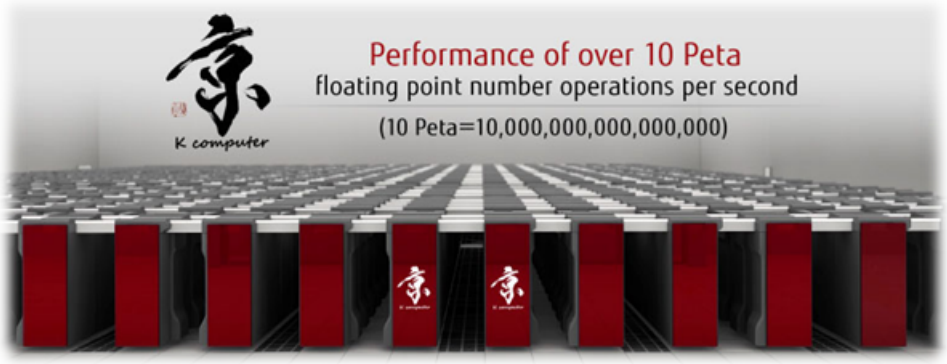


Figure 1.8: K-Computer

K-Computer has been used on number of successful case studies. First, the machine took the first-place rankings in the 2011 HPC Challenge Awards, which considered various benchmarks aiming at testing different hardware capabilities. In addition, astrophysical N-body simulations of one trillion particles were performed on the full system of the K computer and awarded the 2012 ACM Gordon Bell Prize. The 6-dimensional mesh/torus of the K-computer provides an exceptional communication flexibility, which makes it globally efficient on standard applications. As it uses to be with supercomputers, the K-Computer is now open for shared use.

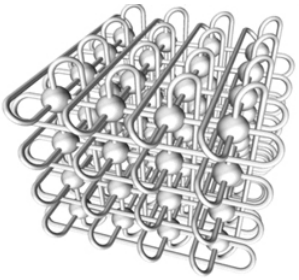


Figure 1.9: TOFU

1.3.4 IBM SuperMUC

SuperMUC is an IBM supercomputer, which was ranked sixth world's fastest supercomputer in the November 2012 top500 ranking. SuperMUC, hosted at the Leibniz Supercomputing Centre "Leibniz-Rechenzentrum" (LRZ / Germany), has delivered a remarkable 2.89 PFlop/s Linpack performance (nearly 3 thousand trillion calculations per second) over a theoretical peak of 3.18 PFlop/s. The close gap between the sustained and the peak performances is clearly a good point for this supercomputer too. The machine is made of 18,432 8-cores Intel Xeon Sandy Bridge-EP processors, thus a total of 147,456 cores clocked at 2.70 GHz each. The overall memory space is 288 TB. Its global power consumption is estimated around 3.42 megawatts, nearly the same *performance/power* ratio as the K-computer (0.84 MFlops/watts). However, SuperMUC uses a revolutionary form of warm water cooling developed by IBM, called "aquasar", which consumes 40% less energy than a comparable air-cooled machine, thus making the system energy efficient. The network is a non-blocking bi-directional tree based on Infiniband FDR, with a point-to-point throughput of 5 GB/s and a latency of 160 nanoseconds.

Figure 1.10: SuperMUC supercomputer

SuperMUC is intended to assist scientists in cutting-edge research in number of fields like geophysics (including prediction of earthquakes), aerodynamics (reduction of aircraft noise), biology (modeling blood flow through and artificial heart). The outstanding efficiency of the machine makes it a good candidate for CPU intensive applications like large-scale simulations in experimental and life sciences, structure calculations in material sciences, linear algebra, to name a few. The high energy efficiency of the system (counting the benefit of the "aquasar" cooling) fits the need of large-scale high throughput computing and offers a good opportunity for a widespread shared use. Last, but the least, the affordable maintenance budget of SuperMUC, due again to its energy efficiency, is a positive when it comes to a context where cost saving is vital, like academic institutions.

1.3.5 Tianhe-1A - NUDT

Tianhe-1A, a hybrid CPU/GPU supercomputer developed at the National university of Defense Technology (NUDT / China), was ranked eighth world's fastest supercomputer in November 2012 top500 ranking, and was the fastest in a 2010 edition. The current update of Tianhe-1A, installed at the National SuperComputer Center in Tianjin (NSCC-TJ / China), has showed a noticeable 2.56 PFlop/s Linpack performance (nearly 2.5 thousand trillion calculations per second) over a theoretical peak of 4.7 PFlop/s. Tianhe-1A is a GPU-based supercomputer, made up with 7168 computing nodes, each featuring two 6-cores Intel Xeon X5670 2.93 GHz, and one NVIDIA M2050 GPU (14 cores), which makes a total of 186, 368 cores. The total memory space is 262 TB, and the power dissipation at full load is 4.04 megawatts. The interconnection topology is an optic-electronic hybrid fat-tree structure with the bi-directional bandwidth of 20 GB/s, and a latency of 1.57 microseconds.

Figure 1.11: Tianhe-1A supercomputer

Among the applications that can benefit from such Tianhe-1A, the National Supercomputer Centre in Tianjin has reported number of successful use cases. First, as China became one of the most oil-importing countries, Tianhe-1A has been considered to provide technological support for the oil exploration so as to enhance the international competitiveness of chinese companies. This was achieved using the GeoEast-lightning software to deal with the two oil seismic exploration data.

Table 1 reports their experimentation from the computation point of view. Other aspects include 3D reverse-time migration, Laplace-Fourier waveform inversion, and large-scale geological studies.

Table 1 GeoEast-lightning software testing result on TH-1A

Surface (sq.km.)	Depth (km.)	Data (shot)	Nodes	Use ratio	Computing time (hour)
1050	5	70000	7100	99%	16
680	9	80000	7000	99%	40

Biology is also concerned, with gene sequencing, prediction of protein structure, and high-throughput virtual screening. The supercomputer is also used for intensive engineering simulation (automotive crash, metal forming, electrical design, rotating machinery, hydraulic structures). We think that codes written to run on Tianhe should be optimized to execute at the fastest to save energy.

1.3.6 The IBM-CELL

The **CELL Boradband Engine** [6, 27] is a multi-core chip that includes nine processing elements. One core, the POWER Processing Element (PPE), is a 64-bit Power Architecture. The remaining eight cores, the Synergistic Processing Elements (SPEs), are Single Instruction Multiple Data (SIMD) engines (3.2GHz) with 128-bit vector registers and 256 KB of local memory, referred to as local store (LS). Figure fig. 1 provides a synthetic view of the CELL architecture [45].

Figure 1.12: IBM CELL BE organization

Programming the CELL is mainly a mixture of single instruction multiple data parallelism, instruction level parallelism and thread-level parallelism. The chip was primarily intended for digital image/video processing, but was immediately considered for general purpose scientific programming (see [40] for an exhaustive report on the potential of the CELL BE for several key scientific computing kernels). A specific consideration for QR factorization is presented in [23]. We have achieved valuable implementation of Lattice Quantum ChromoDynamics (LQCD) kernel [34], the Algebraic Path Problem [33], and the Harris Corner Detection algorithm [35]. Nevertheless, exploiting the capabilities of the CELL in a standard programming context is really challenging. The programmer has to deal with hardware and software constraints like data alignment, local store size, double precision penalty, different level of parallelism. Efficient implementation on the CELL is commonly a conjunction of a good computation/DMA overlap and a heavy use of the SPU intrinsics. Although the potential of the CELL-BE, its hard programmability has clouded the horizon. Consequently, the project was suspended, but a similar architecture is still available on PS3 consoles. Moreover, it is possible that the basic ideas that were used to create the CELL-BE will be found on some future generation accelerated architectures.

1.3.7 Graphic Processing Unit

Graphic processing unit (usually referred to as GPU) is a specialized microprocessor that offloads and accelerates graphics rendering from the central processor [9]. It was primarily a graphics chip, acting as a fixed-function graphics processor. Gradually, the chip became increasingly programmable and computationally powerful, thereby leading to the GPU. Now, GPU is used jointly with the CPU for general-purpose scientific and engineering applications. The highly parallel structure of modern GPUs makes them very efficient than traditional

CPUs for algorithms where processing of large blocks of data can be done in parallel, in addition to classical stream processing applications. This has pushed computer scientist to start thinking about an effective use of GPU to accelerate a wider range of applications, thus leading to the advent of the so-called GPGP (General-Purpose computation on Graphics Processing Units). In GPGPU, a GPU is viewed as a high-performance many-core processor that can be used, under the management of a traditional CPU, to achieve a wide range of computing tasks at a tremendous speed. At the earlier stage of GPGPU, the main concern was how to efficiently exchange data between the CPU and the GPU. This CPU-to-GPU bottleneck [8], often shirked in some very optimistic reports, has been one of the main hurdles on the GPGPU ascent. Another critical point was the severe slowdown on double precision processing, which is essential in cutting-edge numerical studies. These two issues have been seriously addressed in current generation GPUs, thus making them an effective general purpose computing alternative. In certain applications requiring massive vector operations, this can yield several orders of magnitude higher performance than a conventional CPU. Figure 1.13 displays an example of processing time improvement of a GPU over a traditional CPU. This example, taken from the NDVIDIA website, reports a benchmark about solving Navier-Stokes equations on various grid sizes. Other reported success stories are: a 12x speedup on an orthorectification algorithm and a 41x speedup on the pan sharpening process by Digital Globe; a 3x (resp. 5x) speedup on solving a linear system and a 8x speedup on solving second-order wave equation in MATLAB [47, 46]; a 8x speedup on basic linear algebra subroutines (cuBLAS) [49]; to name a few.

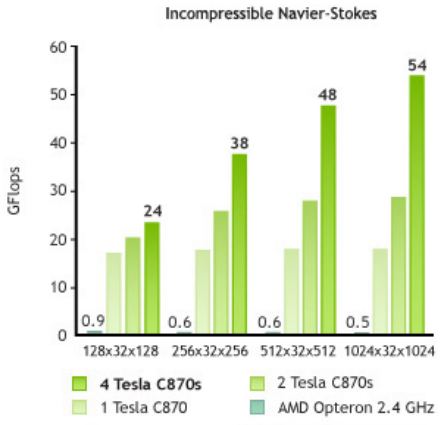
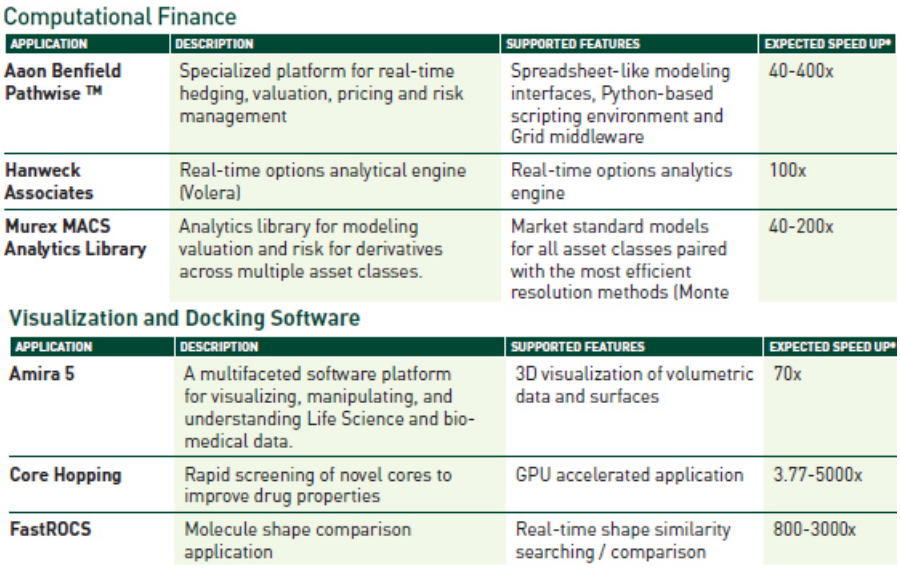


Figure 1.13: Sample GPU speedup

Figure 1.14 provides a selection of potential performances from NVIDIA [48].



Computational Finance

APPLICATION	DESCRIPTION	SUPPORTED FEATURES	EXPECTED SPEED UP*
Aaon Benfield Pathwise ™	Specialized platform for real-time hedging, valuation, pricing and risk management	Spreadsheet-like modeling interfaces, Python-based scripting environment and Grid middleware	40-400x
Hanweck Associates	Real-time options analytical engine (Volera)	Real-time options analytics engine	100x
Murex MACS Analytics Library	Analytics library for modeling valuation and risk for derivatives across multiple asset classes.	Market standard models for all asset classes paired with the most efficient resolution methods (Monte	40-200x

Visualization and Docking Software

APPLICATION	DESCRIPTION	SUPPORTED FEATURES	EXPECTED SPEED UP*
Amira 5	A multifaceted software platform for visualizing, manipulating, and understanding Life Science and bio-medical data.	3D visualization of volumetric data and surfaces	70x
Core Hopping	Rapid screening of novel cores to improve drug properties	GPU accelerated application	3.77-5000x
FastROCS	Molecule shape comparison application	Real-time shape similarity searching / comparison	800-3000x

Figure 1.14: application The use of GPUs to faster the

The use of GPUs to faster the computation is really coming to the vogue, with the hope of saving energy through shorten execution times. This has motivated the consideration of hybrid CPU/GPU supercomputers and the use of GPU a key device in Cloud computing [17]. Another important point when it comes to parallelism among GPUs [15, 31] is data exchanges, which still need to transit via the referent CPU. This problem is also addressed in current and future generations of GPU, with the aim of having a direct cooperation between the GPUs. Figure 1.15 illustrates one aspect of the concept via the so-called dynamic parallelism [33].

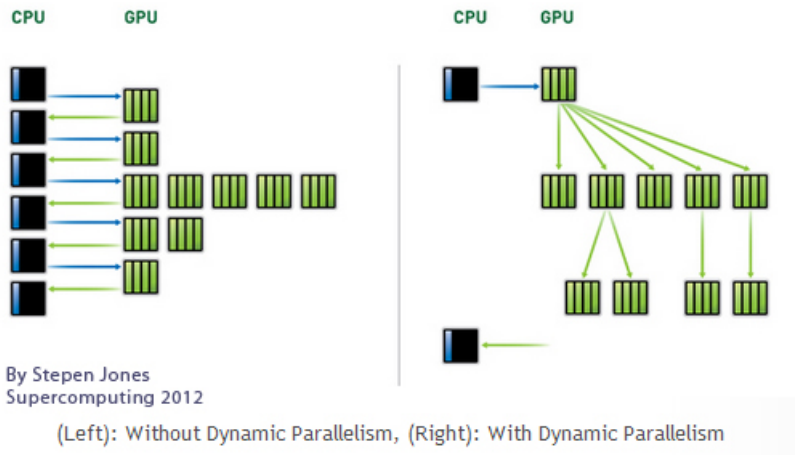


Figure 1.15: Dynamic parallelism with GPUs

1.4 About the interconnect

We have so far focused on the global processing speed that supercomputers can offer to end-users, with an emphasis on the local efficiency of the computing node and how much is there on the machine. Indeed, a supercomputer is made up with several independent computing nodes, but they need to cooperate and exchange data in order to execute a macroscopic task. What we get from there is the so-called sustained performance, which is most of times far from the theoretical peak. In addition to the gap between sustained and peak performances on a node, there is an additional overhead coming from data exchanges between nodes, which is the main concern of the interconnect efficiency. First note that this aspect is not counted when estimating the peak performance, nor external I/O operations. However, depending on the application, data communication can yield a significant impact on the overall performance, thus breaking the scalability on large-scale supercomputers. The special case of applications involving stencil computation is noteworthy. The Lattice Quantum ChromoDynamics (LQCD), the lattice discretized theory of the strong nuclear force, is a nice example with a gigantic number of sites, each of them having 8 neighbors [10]. When two computing nodes have to exchange data, it is well known that this is better done with a direct communication whenever possible; otherwise a slower multi-hop transfer will take place. The concern here is the mismatch between the virtual topology (from the scheduling) and the physical one (from the target machine). The interconnect of a supercomputer should offer a good flexibility for internode communication. The underlying topology should exhibit either high local degrees or shorter internode distances. Figure 1.16 outlines a classical interconnect available on supercomputers.

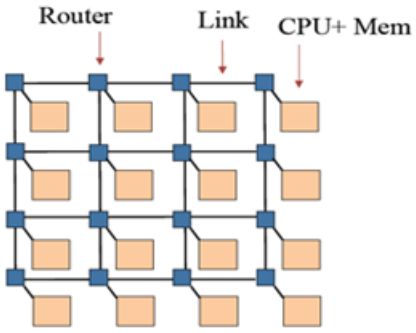


Figure 1.16: Typical supercomputer interconnect

Alongside network topology and bandwidth, communication latency is crucial. The state-of-the-art is around a microsecond, which is acceptable for a point-to-point communication, but less for a multi-hop transfer. Depending on the physical topology and traffic, interprocessor communication might suffer from network congestion, resulting in a significant increase of the sustained latency. Overlapping computation and communication will certainly remain a key ingredient for scalability.

1.5 Trends and future of supercomputers

Processor manufacturers are constantly improving their products by tweaking CPU components and implementing new hardware concepts. The aim is to keep providing increasingly powerful computers for basic issues and large-scale supercomputers for cutting-edge research and engineering. There is a kind of game between progress and need, where we iteratively push the limits and try to go beyond. Harvesting computing cycles for science will certainly change the landscape of experimental research and shorten the path to scientific discovery and technical insights.

As we have so far explained, increasing the (aggregated) processor speed raises number of technical challenges that need to be addressed carefully in order to make their benefit clear to the community. Indeed, the gap between the peak performance and the sustained performance is a genuine concern. This is like gross salary and net salary from the employee viewpoint. Users expect supercomputers to be powerful enough for their applications, not in absolute. Thus, getting close to the maximum performance will be a crucial request. From the hardware point of view, this means number of improvement: memory latency at all hierarchy levels should be reduced; opportunity should be given to the programmer to manage memory features as desired; data exchanges between different memory levels should be improved by adding additional buses; the penalty for accessing distant parts of a NUMA memory should be revisited; the set of vector instructions should be soundly extended; network capability should be improved (topology, bandwidth, and latency) in order to lower enough the communication overhead. At the algorithmic level, the scheduling should be aware of the Amdhal law [21].

The question of heat dissipation and power consumption will sit on top of major concerns. It is possible that, at some points, performance will be sacrificed because of the energy constraint. A typical node of a supercomputer will be made of a traditional multicore processor with several moderate cores, coupled with high-speed accelerator units (mainly GPUs). The idea behind relying on accelerators is that they will be fast enough to significantly reduce the overall execution time, thereby reducing the corresponding heat dissipation. It is important to understand this is a local reasoning, the case of high throughput computation remaining problematic. Indeed, we cannot expect to always compute by spots. Certain kinds of application like simulations, tracking, data assimilation, to name a few, require continuous heavy calculations. The question will be how to keep the benefit of acceleration over a long period of computing time without the punishment of an unacceptable power consumption or hardware failure. Thus, research investigations on the energy efficiency of computing systems will be of a particular interest, both from the hardware side and the programming standpoint. Alongside these efforts, researches on efficient and affordable cooling systems will be also crucial.

Another trend for future innovation, a part from increasing processors horsepower, is the ability to leverage distant power with an increasingly diverse collection of devices. Cloud computing offers a great alternative on mass storage, software and computing devices. Federating available computing resources, assuming sufficiently fast network, is certainly a valuable way to offer a more powerful computing system to the community. The main advantage is that the maintenance cost is mutualized and the users pay only for what they have really consumed. In addition, more related to the Software as a Service (SaaS) feature, users instantly benefit from updates, new releases, and new software. There is also an opportunity to share data and key parameters. This approach of federating available resources can be also seen as a way to save power consumption, as it prevents wastage. The topic of cloud computing is coming to

the vogue and will probably be adopted for major large scale scientific experiments, assuming non sensitive data. The challenge for computer scientist is how to efficiently schedule a given set of tasks on the available set of resources in order to serve the request at the user convenience, while taking care of energy.

From the programming point of view, there are number of serious challenges that need to be addressed or remain under deeper investigations. The heterogeneity of current and upcoming supercomputers requires the use of hybrid codes, which is another level of programming complexity. One might think of using (semi-)automatic code generators, thus concentrate on a higher level abstraction. Programmers will, at certain point, rely on the output of those code generation frameworks, which is not always easy to accept, and otherwise raises a number of practical issues related to debugging, maintenance, adaptability, tuning, and refactoring. Figure 1.17 displays an example of a complex code design framework [11].

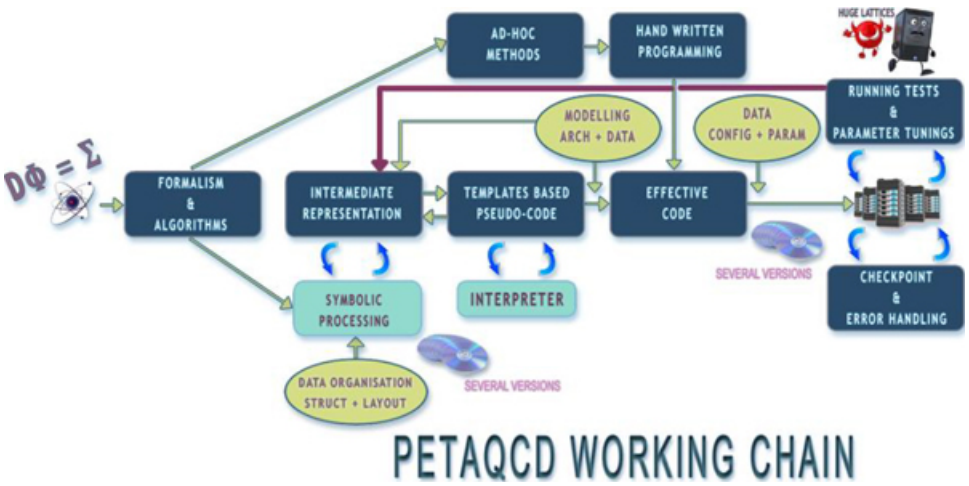


Figure 1.17: Sample hybrid programming chain

As the number of cores is increasing, with various packaging models, scalability will be an important issue for programmers. Some of the considerations that suited for single-threaded code have to be revised when it comes to multi-threaded version. Data locality is one of them, since the so-called false-sharing is also caused by an inappropriate locality. Mixing distributed memory model and shared memory model should become a standard.

Bibliography

[1] . , *Get Connected With Ohm's Law* http://www.tryengineering.org/lessons/ohmslaw.pdf

[2] Bianca Schroeder and Garth A. Gibson, *A Large-Scale Study of Failures in High-Performance Computing Systems*, Proceedings of the International Conference on Dependable Systems and Networks (DSN2006), Philadelphia, PA, USA, pp. 249-258, June 25-28, 2006. http://repository.cmu.edu/pdl/46/

[3] Brock, M., Goscinski, A., *Execution of Compute Intensive Applications on Hybrid Clouds (Case Study with mpiBLAST)*, Sixth International Conference on Complex, Intelligent,and Software Intensive Systems, (CISIS) 2012, Palermo, Italy, July 4-6, 2012. http://www.epcc.ed.ac.uk/wp-content/uploads/2011/11/AleksandarDraganov.pdf

[4] K.W. Cameron and R. Ge, *Predicting and Evaluating Distributed Communication Performance*, Proc. 2004 ACM/IEEE Conf. Supercomputing, IEEE CS Press, 2004.

[5] K.W. Cameron et al., *High-Performance, Power-Aware Distributed Computing for Scientific Applications*, Computer, vol. 38(11), pp.40-47, Nov. 2005,

[6] Cell SDK 3.0. www.ibm.com/developerworks/power/cell

[7] Charles Severance and Kevin Dowd, *High Performance Computing*, online book, Rice University, Houston, Texas, 2012. http://open.umich.edu/sites/default/files/col11136-1.5.pdf

[8] Chris Gregg and Kim Hazelwood, *Where is the Data? Why You Cannot Debate CPU vs. GPU Performance Without the Answer*, International Symposium on Performance Analysis of Systems and Software (ISPASS), Austin, TX. April 2011. http://www.cs.virginia.edu/kim/docs/ispass11.pdf

[9] Chris McClanahan, *History and Evolution of GPU Architecture*, http://mcclanahoochie.com/blog/wp-content/uploads/2011/03/gpu-hist-paper.pdf, 2010.

[10] David Brock, *Understanding Moore's Law: Four Decades of Innovation*, Chemical Heritage Foundation, 2006.

[11] M. Djurfeldt, C. Johansson, . Ekeberg, M. Rehn, M. Lundqvist, and A. Lansner, *Brain-scale simulation of the neocortex on the IBM Blue Gene/L supercomputer*, IBM J. Res. Dev., vol. 52(1-2), pp.31-41, january 2008. http://www.diva-portal.org/smash/get/diva2:220701/FULLTEXT01

[12] M. Djurfeldt, C. Johansson, . Ekeberg, M. Rehn, M. Lundqvist, and A. Lansner, *Massively Parallel Simulation of Brain-Scale Neuronal Network Models*, Technical Report TRITA-NA-P0513, KTH, School of Computer Science and Communication Stockholm, 2005. http://www.diva-portal.org/smash/get/diva2:220701/FULLTEXT01

[13] J. Dongarra and P. Luszczek, Introduction to the HPC Challenge Benchmark Suite, tech. report, Univ. Tennessee, 2004.
www.cs.utk.edu/ luszczek/pubs/hpcc-challenge-intro.pdf.

[14] X. Fan, C.S. Ellis, and A.R. Lebeck, *The Synergy between Power-Aware Memory Systems and Processor Voltage Scaling*, Proc. 3rd Int'l Workshop Power-Aware Computing Systems, LNCS 3164, Springer-Verlag, 2003, pp. 164-179;

[15] Fan, Zhe and Qiu, Feng and Kaufman, Arie and Yoakum-Stover, Suzanne, *GPU Cluster for High Performance Computing*, ACM/IEEE conference on Supercomputing (SC '04), Pittsburgh, November 6-12, 2004.

[16] Frank Wilczek, *What QCD Tells Us About Nature and Why We Should Listen*, Nuc. Phys. A 663, 320, 2000.

[17] G. Giunta, R. Montella, G. Agrillo, G. Coviello, *A GPGPU Transparent Virtualization Component for High Performance Computing Clouds*, 16th International Euro-Par Conference, Ischia, Italia, August 31 - September 3, 2010.

[18] S. Graham, M. Snir, and C. Patterson, eds., Getting Up to Speed: The Future of Supercomputing, National Academies Press, February 2005.

[19] Gupta, Abhishek; Milojicic, Dejan, *Evaluation of HPC Applications on Cloud*,HP Laboratories Technical Report (HPL-2011-132), 2011.
http://www.hpl.hp.com/techreports/2011/HPL-2011-132.pdf

[20] Helias Moritz, Kunkel Susanne, Masumoto Gen, Igarashi Jun, Eppler Jochen Martin, Ishii Shin, Fukai Tomoki, Morrison Abigail, Diesmann Markus, *Supercomputers ready for use as discovery machines for neuroscience*, Frontiers in Neuroinformatics, vol. 6(26), 2012.
http://www.frontiersin.org/Neuroinformatics/10.3389/fninf.2012.00026/abstract

[21] M. Hill and M. Marty, *Amdahl?s law in the multicore era*, Computer, vol. 41, no. 7, pp. 33 ?38, 2008.

[22] Hsu, Chung-hsing and Feng, Wu-chun, *A Power-Aware Run-Time System for High-Performance Computing*, ACM/IEEE conference on Supercomputing (SC '05), Seattle, November 12-18, 2005.

[23] Jakub Kurzak and Jack Dongarra, *QR factorization for the Cell Broadband Engine*, Scientific Programming, vol. 17(1-2), P. 31-42, 2009.

[24] S. Kumar, *Optimization of All-to-All Communication on the Blue Gene/L Supercomputer*, 37th International Conference on Parallel Processing (ICPP'08), Portland, Oregon, USA, September 8?12, 2008.

[25] Michael A. Bender, David P. Bunde, Erik D. Demaine, Sndor P. Fekete, Vitus J. Leung, Henk Meijer, Cynthia A. Phillips, *Communication-Aware Processor Allocation for Supercomputers*, 9th International Workshop, WADS 2005, Waterloo, Canada, August 15-17, 2005.

[26] Nagel, Wolfgang E.; Krner, Dietmar B.; Resch, Michael M (Eds.), *High Performance Computing in Science and Engineering '12*, Springer Book Archives,, 2013.

[27] H. Peter Hofstee, *Power Efficient Processor Design and the Cell Processor*, http://www.hpcaconf.org/hpca11/slides/Cell_Public_Hofstee.pdf.

[28] Peter Kogge et al., *ExaScale Computing Study: Technology Challenges in Achieving Exascale Systems*, DARPA report, 2008.

`http://users.ece.gatech.edu/mrichard/ExascaleComputingStudyReports/exascale_final_report_100208.pdf`

[29] Rajesh Bordawekar, Uday Bondhugula, Ravi Rao, *Can CPUs Match GPUs on Performance with Productivity?: Experiences with Optimizing a FLOP-intensive Application on CPUs and GPU*, Research Report RC25033, IBM T.J. Watson Research Center, 2010.

[30] Rajkumar Buyya , *High Performance Cluster Computing: Architectures and Systems* (volume 1 volume 2), Prentice Hall, 1999.

[31] S. Ryoo, C. Rodrigues, S. Stone, S. Baghsorkhi, S.-Z. Ueng, and W. mei Hwu, *Program Optimization Study on a 128-Core GPU*, Workshop on General Purpose Processing on Graphics Processing, 2008.

[32] Shuai Che, Michael Boyer, Jiayuan Meng, David Tarjan, Jeremy W. Sheaffer, Sang-Ha Lee and Kevin Skadron, *Rodinia: A benchmark suite for heterogeneous computing*, IEEE International Symposium on Workload Characterization (IISWC), Oct. 2009.

[33] Stephen Jones, *Inside the Kepler Architecture*, Supercomputing (SC12), Salt Lake City, USA, November 10-16, 2012.

[34] C. Tadonki, G. Grosdidier, and O. Pene, *An efficient CELL library for Lattice Quantum Chromodynamics*, International Workshop on Highly Efficient Accelerators and Reconfigurable Technologies (HEART) in conjunction with the 24th ACM International Conference on Supercomputing (ICS), pp. 67-71, Epochal Tsukuba, Tsukuba, Japan, June 1-4, 2010.

[35] C. Tadonki, L. Lacassagne, E. Dadi, M. Daoudi, *Accelerator-based implementation of the Harris algorithm*, 5th International Conference on Image Processing (ICISP 2012), Agadir, Maroc, June 28-30, 2012.

[36] C. Tadonki, *Ring pipelined algorithm for the algebraic path problem on the CELL Broadband Engine*, Workshop on Applications for Multi and Many Core Architectures (WAMMCA 2010) in conjunction with the International Symposium on Computer Architecture and High Performance Computing , Petropolis, Rio de Janeiro, Brazil, October 27-30, 2010.

[37] Thomas Lippert, *Lattice Quantum Chromodynamics and the Impact of Supercomputing on Theoretical High Energy Physics*, DEISA Symposium, Paris, May 9-10, 2005.
`http://www.deisa.eu/news_press/symposium/Paris2005/presentations/lippert_DEISA_2005.pdf`

[38] Tomas Oppelstrup, *Introduction to GPU programming*, PDC summer school 2008.
`http://www.training.prace-ri.eu/uploads/tx_pracetmo/gpu-talk-pdc.pdf`

[39] Victor W Lee, Changkyu Kim, Jatin Chhugani, Michael Deisher, Daehyun Kim?, Anthony D. Nguyen?, Nadathur Satish?, Mikhail Smelyanskiy, Srinivas Chennupaty., Per Hammarlund., Ronak Singhal. and Pradeep Dubey, Debunking the 100X GPU vs. CPU Myth: An Evaluation of Throughput Computing on CPU and GPU, ISCA?10, June 19?23, Saint-Malo, France, 2010.
`http://pcl.intel-research.net/publications/isca319-lee.pdf`

[40] S. Williams, J. Shalf, L. Oliker, S. Kamil, P. Husbands, and K. Yelick, *Scientific Computing Kernels on the Cell Processor*, International Journal of Parallel Programming, 2007.

[41] Xiang Rao, Huaimin Wang, Dianxi Shi, Zhenbang Chen, Hua Cai, Qi Zhou, Tingtao Sun, *Identifying faults in large-scale distributed systems by filtering noisy error logs*, IEEE/IFIP 41st International Conference on Dependable Systems and Networks Workshops, pp.140-145, Hong Kong, China, June 27-30, 2011.

[42] Zhou Zhou, Wei Tang, Zhiling Lan, Ziming Zheng, Desai, N., *Evaluating Performance Impacts of Delayed Failure Repairing on Large-Scale Systems*, Proceedings of the 2011 IEEE International Conference on Cluster Computing (CLUSTER'11), pp.532-536, 2011.

[43] *Enhanced Intel SpeedStep Technology for the Intel Pentium Processor* `ftp://download.intel.com/design/network/papers/30117401.pdf`, white paper, march 2004.

[44] *Performance Analysis of High Performance Computing Applications on the Amazon Web Services Cloud*, Proceedings of the 2010 IEEE Second International Conference on Cloud Computing Technology and Science, pp. 159-168, 2010.

[45] `http://fr.wikipedia.org/wiki/Cell`

[46] `http://www.mathworks.fr/company/newsletters/articles/gpu-programming-in-matlab.html`

[47] `http://www.mathworks.fr/products/demos/shipping/distcomp/paralleldemo_gpu_backslash.html`

[48] `http://www.nvidia.com/docs/IO/123576/nv-applications-catalog-lowres.pdf`

[49] `https://developer.nvidia.com/cublas`

[50] `https://www.petaqcd.org`

Chapter 2

Large-scale optimization

2.1 Foundations and background

Operations research is the science of decision making. The goal is to derive suitable mathematical models for practical problems and study effective methods to solve them as efficient as possible. For this purpose, *mathematical programming* has emerged as a strong formalism for major problems. Nowadays, due to the increasing size of the market and the pervasiveness of network services, industrial productivity and customers services should scale up with a whooping need and a higher quality requirement. In addition, the interaction between business operators has reached a noticeable level of complexity. Consequently, for well established companies, dealing with optimal decisions is critical to survive, and the key to achieve this purpose is to exploit recent operation research advances. The objective is to give a quick and accurate answer to practical instances of critical decision problems. The role of operation research is also central in cutting-edge scientific investigations and technical achievements. A nice example is the application of the *traveling salesman problem* (TSP) on *logistics*, *genome sequencing*, *X-Ray crystallography*, and *microchips manufacturing*[5]. Many other examples can be found in real-world applications[108]. A nice introduction of combinatorial optimization and complexity can be found in [105, 39].

The noteworthy increase of supercomputers capability has boosted the enthusiasm for solving large-scale combinatorial problems. However, we still need powerful methods to tackle those problems, and afterward provide efficient implementation on modern computing systems. We really need to seat far beyond brute force or had hoc (unless genius) approaches, as increasingly bigger instances are under genuine consideration. Figure 2.1 displays an overview of a typical workflow when it comes to solving optimization problems.

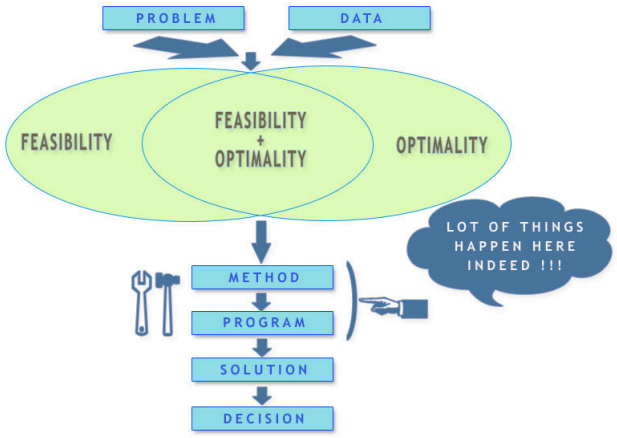


Figure 2.1: Typical operation research workflow

Most of common combinatorial problems can be written in the following form

$$\begin{cases} \text{minimize} & F(x) \\ \text{subject to} & P(x) \\ & x \in S, \end{cases} \tag{2.1}$$

where F is a polynomial, $P(x)$ a predicate, and S the working set, generally $\{0,1\}^n$ or $\mathcal{Z}^n$. The *predicate* is generally referred to as *feasibility constraint*, while F is the known as the

objective function. In the case of a pure feasibility problem, F could be assumed to be constant. An important class of optimization problem involves a linear objective function and linear constraints, thus the following generic formulation

$$\begin{cases} \text{minimize} & c^T x \\ \text{subject to} & Ax \leq b \\ & x \in \mathbb{Z}^p \times \mathbb{R}^{n-p}, \end{cases} \tag{2.2}$$

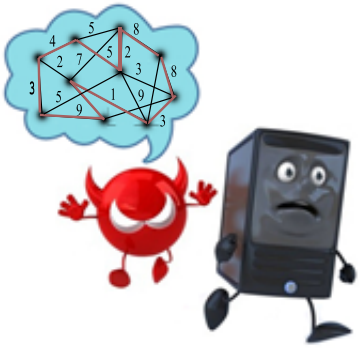
where $A \in \mathbb{R}^{m \times n}$, $c \in \mathbb{R}^n$, $b \in \mathbb{R}^m$, and $p \in \{0, 1, \cdots, n\}$. If $p = 0$ (resp. $p = n$), then we have a so-called *linear program* (resp. *integer linear program*), otherwise we have a *mixed integer program.* The corresponding acronyms are LP, ILP, and MIP respectively. In most cases, integer variables are *binary* $0 - 1$ *variables.* Such variables are generally used to indicate a choice. Besides linear objective functions, quadratic ones are also common, with a *quadratic term* proportional to $x^t Q x$. We now state some illustrative examples.

Example 1 The Knapsack Problem (KP)*[79]. The Knapsack Problem is the problem of choosing a subset of items such that the corresponding profit sum is maximized without having the weight sum to exceed a given capacity limit. For each item type i, either we are allowed to pick up at most* 1 *(binary knapsack)[35], or at most m_i (bounded knapsack), or whatever quantity (unbounded knapsack). The bounded case may be formulated as follows(2.3):*

$$\begin{cases} \textit{maximize} & \sum_{i=1}^{n} p_i x_i \\ \textit{subject to} & \\ & \sum_{i=1}^{n} w_i x_i \leq c \\ & x_i \leq m_i \quad i = 1, 2, \cdots, n \\ & x \in \{0,1\}^{n \times n} \end{cases} \tag{2.3}$$

Example 2 The Traveling Salesman Problem (TSP)*[5]. Given a valuated graph, the Traveling Salesman Problem is to find a minimum cost cycle that crosses each node exactly once (tour). Without lost of generality, we can assume positive cost for every arc and a zero cost for every disconnected pair of vertices. We formulated the problem as selecting a set of arcs (i.e. $x_{ij} \in \{0,1\}$) so as to have a tour with a minimum cost(2.4). Understanding how the way constraints are formulated implies a tour is left as an exercise for the reader.*

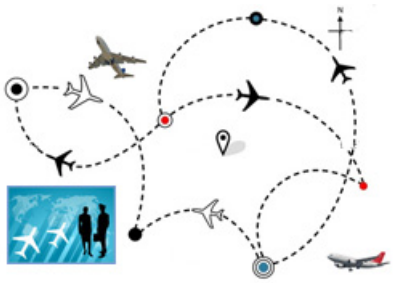
$$\begin{cases} \textit{minimize} & \sum_{i=1}^{n} \sum_{j=1}^{n} c_{ij} x_{ij} \\ \textit{subject to} & \\ & \sum_{j=1}^{n} x_{ij} = 1 \quad i = 1, \cdots, n, i \neq j \\ & \sum_{i=1}^{n} x_{ij} = 1 \quad j = 1, \cdots, n, i \neq j \\ & x \in \{0,1\}^{n \times n} \end{cases} \tag{2.4}$$



The TSP has an a priori $n!$ complexity. Solving any instance with $n = 25$ using the current world fastest supercomputer (TITAN-CRAY XK7) might require 25 *years of calculations.*

Example 3 The Airline Crew Pairing Problem (ACPP)*[125]. The objective of the ACPP is to find a minimum cost assignment of flight crews to a given flight schedule. The problem can be formulated as a set partitioning problem(2.5).*

$$\begin{cases} \textit{minimize} & c^T x \\ \textit{subject to} & Ax = 1 \\ & x \in \{0,1\}^n \end{cases} \tag{2.5}$$



In equation (2.5), each row of A represents a flight leg, while each column represents a feasible pairing. Thus, a_{ij} tells whether or not flight i belongs to pairing j.

In practice, the feasibility constraint is mostly the heart of the problem. This the case for the TSP, where the feasibility itself is a difficult problem (the *Hamiltonian cycle*). However, there are also notorious cases where the dimension of the search space S (i.e. n) is too large to be handled explicitly when evaluating the objective function. This is the case of the ACPP, where the number of valid pairings is too large to be included into the objective function in one time. We clearly see that we can either focus on the *constraints* or on *components* of the objective function. In both cases, the basic idea is to get rid of the part that makes the problem difficult, and then reintroduce it progressively following a given strategy. Combined with the well known *branch-and-bound* paradigm[26], these two approaches have led to well studied variants named *branch-and-cut*[124] and *branch-and-price*[12] respectively.

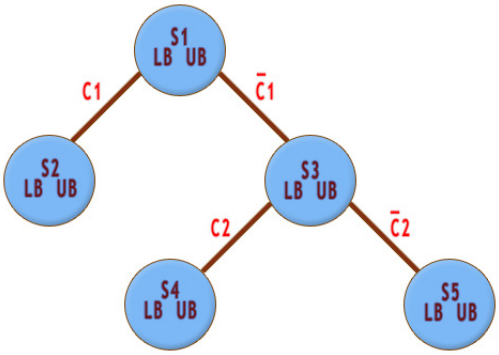


Figure 2.2: Branch-and-bound overview

The key ingredient of this connection between discrete and continuous optimization is *linear programming* (LP). Indeed, applying a *linear relaxation* on the exact formulation of a combinatorial problem, which means assuming continuous variables in place of integer variables, generally leads to an LP formulation from which lower bounds can be obtained (upper bounds are obtained on feasible guests, mostly obtained through heuristics). LP is also used to handle the set of constraints in a *branch-and-cut*, or to guide the choice of new components (*columns*) of the objective function in the *branch-and-price* scheme. Figure 2.3 depicts the linear relaxation of an IP configuration, while Figure 2.4 provides a sample snapshot of an LP driven branch-and-bound.

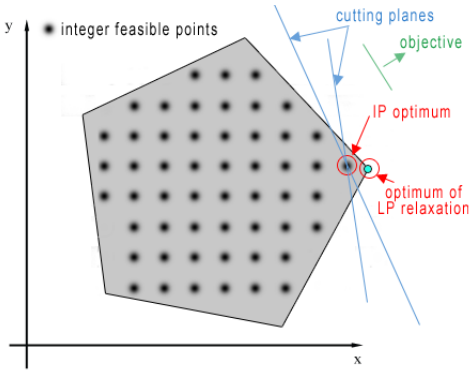


Figure 2.3: Interger programming & LP

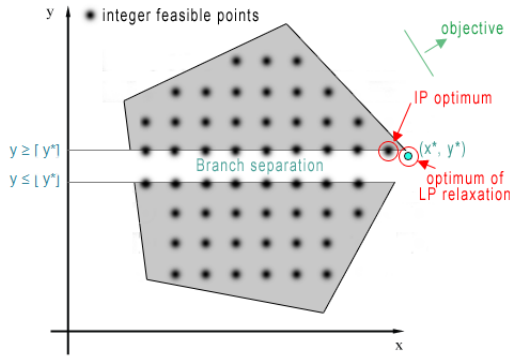


Figure 2.4: Branch-and-bound & LP

Linear programming has been intensively studied and has reached a very mature state, even from the computing standpoint. Nowadays, very large scale LP can now be routinely solved using specialized software packages like CPLEX[133] or MOSEK[135].

Branch-and-bound and its variants can be applied to a mixed integer programming formulation by means of basic techniques like *Bender decomposition*[17] or *Lagrangian relaxation*[87]. Figure 2.5 depicts the basic idea behind these two approaches.

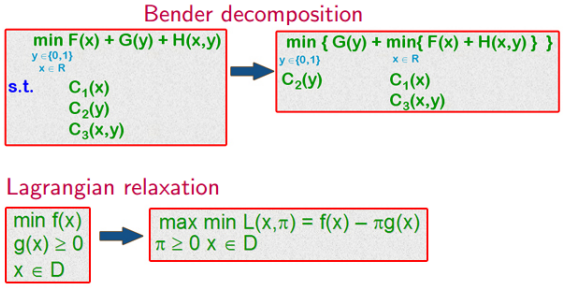


Figure 2.5: Bender decomposition & Langrangian relaxation

The later is likely to yield *non-differentiable optimization* (NDO) problems. Several approaches for NDO are described in the literature[65], including an oracle-based approach[7], which we will later describe in details as it illustrates our major contribution on that topic. Figure 2.6 gives an overview of an oracle based mechanism.

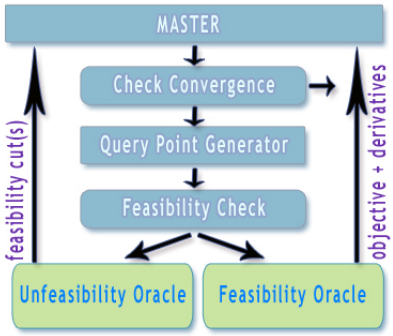


Figure 2.6: Oracle based optimization workflow

From the methodological point of view, optimization (both continuous and combinatorial) has been so far subject to intensive and fruitful investigations. New optimization paradigms or improved variants of classical techniques have reached an acceptable level of maturity, and have proved successful on number of notorious practical problems. However, in some cases, the expected level of performance can be achieved only through parallel implementation on large-scale supercomputers, especially with intractable (but pertinent) combinatorial problems. The idea is to combine the advantages of mathematically powerful algorithms with the capability of machines that have several processors. The existence of commercial multiprocessor computers has created substantial interest in exploring the use of parallel processing for solving optimization problems even for basic issues. The challenge is to find a suitable way to implement the aforementioned techniques (somehow irregular) on modern supercomputers (mostly tailored for regular computations) with an acceptable efficiency. We now provide technical details on how this can be tackled and what has been done.

2.2 Parallel optimization

First, computational complexity studies the intrinsic difficulty of optimization problems and decides which of these problems are likely to be tractable. There is an important set of common problems that can be solved in polynomial time. However, as some of them are recursively solved to get the solution of more difficult problems, improvements are still expected whenever it remains technically possible. A good example of this is the *shortest paths problem*, which appears as a subproblem when solving the *multicommody flow problem*[9]. Basically, we may distinguish *continuous optimization* and *discrete optimization*. However, many approaches in combinatorial optimization have developed a bridge between the discrete universe and the continuous universe through geometric, analytic, and algebraic techniques such as *global optimization*, *semidefinite programming*, and *spectral theory*. Mixed integer programming formulations involve differentiable and/or non differentiable objective functions. Non differentiable case generally comes from the use of *Lagrangian relaxation*, which brings part of the constraints (usually the harder ones) into the objective function. Moreover, the pursuit of efficient algorithms for common combinatorial problems has lead to useful connections among problems and their solutions. As consequence, there is a set of reference optimization problems for which improved solutions are continuously tracked by researchers. For this purpose, *parallel computing* applied to the previously mentioned optimization paradigms is clearly the way to go.

Most of discrete optimization problems are NP- complete[61]; hence their time complexity increases exponentially for all known algorithms. Consequently, parallel processing cannot achieve polynomial complexity without using at least an exponential number of processors. However, the average-time complexity of heuristics and suboptimal algorithms for a wide range of problems are polynomial[82, 127]. Significant advances have been made in the use of powerful heuristics and parallel processing to solve large scale discrete optimization problems. Number of problem instances that were considered computationally intractable on sequential machines are routinely solved on server-class symmetric multi-processors and workstation clusters[71, 121]. There are mainly two research directions in parallel optimization: numerical approaches and non numerical paradigms.

We get a direct impact of parallel computing in numerical optimization through the advances in parallel numerical algebra[29, 52, 49], with some of them being implemented into

effective frameworks[3, 11, 2, 25, 122]. Encapsulating parallel linear algebra routines into optimization codes [43, 114] is a nice way to provide their power to the users without an additional effort. This is still a very critical and challenging topic since parallelizing the linear algebra kernels of optimization algorithms is not an easy task. In fact, matrix factorization updating process in quasi-Newton methods or active set strategies involves vector-vector operations that are not easy to parallelize efficiently [41]. According to Schnabel[112], parallel computing can be performed in numerical optimization through three levels:

- ◇ parallelization of the function and/or the derivative evaluations;
- ◇ parallelization of the linear algebra kernels;
- ◇ modifications of the basic algorithms in order to increase the degree of parallelism.

For the first two levels, a number of solutions have been developed in the last two decades [106, 107, 54, 69]. For most of the interior point (IP) methods for linear programming (LP), quadratic programming (QP), and nonlinear programming, the kernel is the solution of a special linear system [4, 19]. As the iterates approach the boundary of the feasible set or the optimal solution, the system becomes more and more ill-conditioned. Suitable strategies have been developed within a modified Cholesky factorization framework and successfully used in specialized codes as CPLEX[133] and LOQO[134]. Thus, efficient parallel versions of these strategies are very desirable, but challenging, especially for sparse systems. The paper of Durazzi and Ruggiero [50] presents a parallel approximated IP method for QP, based on a preconditioned Conjugated Gradient algorithm. D'Apuzzo and Marino [41] have proposed a parallel Potential Reduction algorithm for the convex quadratic bound constrained problem. A parallel decomposition approach is considered by Zanghirati and Zanni [130] for large scale QPs. Blomwall [23] has proposed a parallel implementation of a Riccati-based primal IP algorithm for multistage stochastic programming.

Regarding the third level, the multidirectional search strategies [89] provide a high level parallelism which can be exploited through a concurrent execution of the minimization processes. Ad hoc or application specific algorithms are also concerned, particularly when large-scale instances are considered [31, 81]. Another case study in statistical model selection is analyzed by Gatu and Kontoghiorrghes [62]. As many fields of numerical analysis, number of algorithms in numerical optimization have been revisited because of parallelism considerations. In [56], approaches to expose parallelism through appropriate partitioning of mathematical programs are reported. Interior point strategies, because of their direct possibility of parallel implementation [42, 73], have received much attention compare to active set algorithms, and have stimulated intensive researches in order to understand and overcome their weak scaling on large supercomputers. Developments in object oriented software for coding and tuning linear algebra algorithms at a high level of abstraction are available in [123, 128]

As previously said, many techniques have so far been developed to provide a link between continuous and discrete formulations. Major recent successes based on such approaches include IP methods for discrete problems, the Goemans-Williamson relaxation of the maximum cut problem, the Chvatal cuts for the traveling salesman problem, and the Gilbert-Pollak's conjecture, to name a few. Parallel algorithms for discrete optimization problems can be obtained in many different ways including the classical *domain decomposition*. SPMD (Single Program Multiple Data) parallelization attempts to enlarge the exploration of the solution space by initiating multiple simultaneous searches towards the optimal solution. These approaches are well implemented by clustering methods. Byrd et al. [31, 30] and Smith and

Schnabel[115] have developed several parallel implementations of the clustering method. Parallelization of classical paradigms have also been explored: *parallel dynamic programming*[63], *branch and bound*[26, 45], *tabu search, simulated annealing*, and *genetic algorithms*[109]. In the paper of Clementi, Rolim, and Urland [37], randomized parallel algorithms are studied for *shortest paths, maximum flows, maximum independent set*, and *matching problems.* A survey of parallel search techniques for discrete optimization problems are presented in [71]. The most active topics are those involved with searching over trees, mainly the *depth-first* and the *best-first* techniques and their variants. The use of parallel search algorithms in games implementation has been particularly successful, the case of IBM's Deep Blue [113] is illustrative.

We now present one of our main contribution in the field of nondiffrentiable convex optimization. As we have explained, such a contribution, made of a method and the corresponding framework, is very useful for both continuous optimization and combinatorial optimization.

2.3 Preamble

The work we are going to describe here is a substantial part of our achievement during a postdoctoral stay at the university of Geneva (LOGILAB). The main content of the ongoing report was published in [7]. Oracle Based Optimization (OBO) conveniently designates an approach to handle a class of convex optimization problems in which the information pertaining to the function to be minimized and/or to the feasible set takes the form of a linear outer approximation revealed by an oracle. Five representative examples are introduced to illustrate the effectiveness of the method at different levels of the optimization process. An efficient algorithm, Proximal-ACCPM, is presented to trigger the OBO approach. Numerical results for the set of selected examples are provided to illustrate the behavior of the method. This paper summarizes several contributions with the OBO approach and aims to give, in a single report, enough information on the method and its implementation to facilitate new applications. On top of this main contribution to convex optimization, we have studied number of illustrative cases [118, 117, 16].

2.4 Introduction

Oracle Based Optimization (OBO) conveniently designates an approach to handle a class of convex optimization problems in which the information pertaining to the function to be minimized and/or to the feasible set takes the form of a linear outer approximation revealed by an oracle. By oracle, we mean a black-box scheme that returns appropriate information on the problem at so-called query points. In convex unconstrained optimization, this information takes the form of a linear support for the epigraph set of the function to be minimized. This class of problems is known as "Nondifferentiable Convex Optimization". We use the terminology OBO to emphasize the principle of the method — a dialog between an optimizer and an oracle — and the fact that we can handle more general classes of problems.

The goal of this report is two-fold. We first intend to present an efficient method, Proximal-ACCPM, that implements an OBO approach. We give a concise but accurate description of the analytic center cutting plane method (ACCPM), and more precisely of its recent enhancements that include a proximal term (Proximal-ACCPM) and a logarithmic barrier on the epigraph of the smooth component of the objective function. The main issue in a cutting plane method is to decide where to query the oracle in order to improve a current polyhedral approximation of the problem. Proximal-ACCPM selects the analytic center of this polyhedral set, that is, the point that minimizes the logarithmic barrier function on that set, augmented with a proximal term. This choice is efficient since it usually requires relatively few query points to achieve an accurate approximation of an optimal solution. Proximal-ACCPM relies on the interior-point methodology to compute the query points. This methodology is well suited to handle non-linear information and makes it easy to implement the extensions we discuss in this report.

Our second goal is to provide a set of application problems that are very different in nature and thus illustrate the versatility of the method. This choice does not cover the full range of applications successfully handled with Proximal-ACCPM. Yet it gives a flavor of what can be done and hopefully it will convince readers to develop applications of their own.

In this work we do not deal with the convergence issue. The pseudo-polynomial complexity of the method on the feasibility problem has been proved in [68, 102] and straightforwardly

extends to optimality problems by casting the latter in the format of a pure feasibility problem. The proofs are involved but the principles underlying the method are relatively simple. Neither will we review the literature on nondifferentiable convex optimization. The field is large and we content ourselves with referring to survey papers [88, 65]. In this presentation we concentrate on the description of the method with some recent extensions and we illustrate its implementation and performance on five large-scale applications recently reported in the literature.

The report is organized as follows. In Section 4 we present the framework of Oracle Base Optimization. Section 5 provides a succinct description of Proximal-ACCPM. Two enhancements of the method are discussed. None of them is really new, but we believe that they crucially contribute to the overall efficiency of the implementation. We also discuss how to compute a lower bound and thus obtain a reliable stopping criterion. Section 6 deals with five illustrative examples. The first one, the well-known multicommodity flow problem, is representative of large-scale continuous optimization. The method has been applied to the linear [8] and the nonlinear [6] cases. The nonlinear version of the multicommodity flow problems we present here is particularly interesting, because part of the problem structure need not be revealed by a first-order oracle. As it is presented in Section 5, Proximal-ACCPM directly incorporates the non-linear information and thus achieves a significant gain of efficiency.

The second application is the p-median problem, a combinatorial optimization problem that is solved by Lagrangian relaxation. This example illustrate how powerful is Lagrangian relaxation to generate lower bounds for the optimal value of this combinatorial problem. These bounds are further used in an enumerative scheme which computes an optimal integer solution. In the same subsection we present the new concept of semi-Lagrangian relaxation, recently introduced in [16]. There, it is shown that using semi-Lagrangian relaxation permits us to solve to optimality the original combinatorial problem without resorting to an enumerative scheme.

The third application deals with air quality control in urban regions and the coupling of modules in Integrated Assessment Models (IAM). The economic activity creates pollutant emissions that are spatially distributed. Geographic and climate models translate those primary pollutant emissions into ozone concentrations which determine air quality. The objective of the study is to find an optimal adjustment of the economic activity that results in acceptable ozone concentrations. The modeling approach consists in coupling two models, a techno-economic model and a climate model, to properly handle the interaction between the economic activity and the air quality. From a methodological point of view, this approach is original as it allows the coupling of two models that have totally different natures.

The fourth application is related to the general topic of *Data Mining*. We address the case of the *linear separation*, which can be solved directly from its native formulation as a liner or quadratic programming problem. We provide an alternative formulation more suitable for very large instances and show that our method performs better than pure LP or QP approach.

Our last case study is based on a general formulation of the *Portfolio Selection Problem*. This a complete solution for a MIP problem for which we have enhanced our solver with a basic implementation of a branch-and-bound mechanism.

The framework can be download at
`https://projects.coin-or.org/OBOE`

2.5 Oracle based optimization

Oracle based optimization deals with the convex programming problem

$$\min\{f(u) = f_1(u) + f_2(u) \mid u \in U \subset \mathbb{R}^n\}, \tag{2.6}$$

where f_1 is a convex function, f_2 is a twice differentiable convex function and U is a convex set. We assume that $f_1(u)$ and U are revealed by a first order oracle while $f_2(u)$ is accessed through a second order oracle in an explicit way. By oracle, we mean a black-box procedure which at any *query point* u returns the information described in Definitions 1 and 2.5 below.

Definition 1 *A first-order oracle for problem (2.6) is a black box procedure with the following property. When queried at u, the oracle returns 1 or 2.*

1. *$u \notin U$ and (a, c) is such that $a^T u' - c \leq 0, \forall u' \in U$ (feasibility cut). In that case, we set $f_1(u) = +\infty$.*
2. *$u \in U$ and (a, c) is such that $a^T u' - c \leq f_1(u'), \forall u' \in U$ (optimality cut). In general, $a \in \partial f_1(u)$, $c = a^T u - f_1(u)$, but this is not necessarily so. The cut may have no intersection with the epigraph set (i.e., may be situated strictly below that set).*

Definition 2 *A second-order oracle for problem (2.6) is a black-box procedure with the following property. When queried at u, the oracle returns the function value and the first and second derivatives of $f_2(u)$.*

In the traditional OBO approach, the function f_2 is handled in the same way as f_1, that is by means of a first-order oracle. This approach looses information. In this work, we exploit the explicit knowledge of the function f_2 and its derivatives in the form of a barrier on the epigraph set.

Assumption 1 *The function f_2 is such that the logarithmic barrier $-\log(\zeta - f_2(u))$ on the epigraph set of f_2, $\{(u, \zeta) \mid \zeta \geq f_2(u),\ u \in U\}$, is self-concordant.*

Remark 1 *The concept of self-concordant function has been introduced by Nesterov and Nemirovski [100] to extend the theory of interior-point methods for linear programming to a more general class of functions. The condition links the second and third derivatives of the function. For a thorough but more readable presentation of the theory of self-concordant functions we refer to [103].*

In many applications, the objective function f_1 is a strictly positively weighted sum of p nonsmooth convex functions

$$f_1(u) = \sum_{i=1}^{p} \pi_i f_{1i}(u).$$

In that expression, we can consider that $f_1(u)$ is revealed by p independent first-order oracles. The epigraph of the function f is the set defined by $\{(u, z, \zeta) \mid \pi^T z \geq f_1(u),\ \zeta \geq f_2(u)\}$. Using this property, problem (2.6) can also be written in as

$$\begin{array}{ll} \min & \pi^T z + \zeta \\ \text{s.t.} & f_{1j}(u) - z_j \leq 0, \quad j = 1, \ldots, p, \\ & f_2(u) - \zeta \leq 0, \\ & u \in U. \end{array} \tag{2.7}$$

This formulation is conveniently named the *disaggregate mode*.

The first order oracle is used to build a polyhedral approximation of the epigraph of f_1. Suppose the oracle has been queried at $u^k,\ k = 1, \ldots, \kappa$, and has returned feasibility and/or optimality cuts associated with those points. The corresponding inequalities are collected in

$$A^T u - E^T z \leq c.$$

In that definition, the subgradients a of the function f_1 form the matrix A while E is a binary matrix that is constructed as follows. If the objective f_1 is treated in an aggregate mode ($p = 1$), then E is a binary row vector. An entry one in E indicates that the z variable is present in the cut, implying that the cut is an *optimality cut*. In contrast, a zero indicates that the cut is a *feasibility cut*. If the objective f_1 is disaggregated into p components, row j of E corresponds to a variable z_j and each column corresponds to a cut. An entry one in row j and column k indicates that the cut k is an optimality cut for $f_{1j}(u)$. If column k is a null vector, then cut k is a feasibility cut.

Let $\bar{\theta}$ be the best recorded value such that $\bar{\theta} = \min_{k \leq \kappa}\{f_1(u^k) + f_2(u^k)\}$. In view of the above definitions, we can define the localization set $\mathcal{L}_\kappa$ as

$$\mathcal{L}_\kappa = \left\{(u, z, \zeta) \mid A^T u - E^T z \leq c,\ f_2(u) \leq \zeta,\ \pi^T z + \zeta \leq \bar{\theta}\right\},$$

which is a subset of an outer approximation of the epigraph of f that contains all optimal pairs $(u^*, f(u^*))$. Thus, the search for a new query point should be confined to the localization set. Among possible solution methods for (2.6), we briefly sketch cutting plane schemes which work as follows:

1. Select a query point in the localization set.

2. Send the query point to the first order oracle and get back the optimality/feasible cuts.

3. Send the query point to the second order oracle to compute the objective function f_2.

4. Update the lower and upper bounds and the localization set.

5. Test termination.

The main issue in the design of a cutting plane scheme is step 1. Different choices lead to different results. We propose a particular method, named *Proximal-ACCPM*, that selects the analytic center of the localization set as the new query point.

2.6 Proximal-ACCPM

It is well-known that efficient methods for non differentiable convex optimization rely on some regularization scheme to select the query point. We discuss here such a scheme; it is based on the concept of proximal analytic center which corresponds to the minimum of the standard logarithmic barrier augmented with a proximal term.

2.6.1 Proximal analytic center

We associate with the localization set a standard (weighted) logarithmic barrier

$$F(s_0, s, \sigma) = -w_0 \log s_0 - \sum_{i=1}^{\kappa} w_i \log s_i - \omega \log \sigma, \tag{2.8}$$

with $(s_0, s, \sigma) > 0$ defined by

$$\begin{array}{rcl} s_0 & = & \bar{\theta} - \pi^T z - \zeta, \\ s_i & = & c_i - (A^T u - E^T z)_i, \quad i \in K = \{1, \ldots, \kappa\}, \\ \sigma & = & \zeta - f_2(u). \end{array}$$

The barrier function is augmented with a proximal term to yield the augmented barrier

$$\Psi(u, s_0, s, \sigma) = \frac{\rho}{2}||u - \bar{u}||^2 + F(s_0, s, \sigma), \tag{2.9}$$

where $\bar{u} \in \mathbb{R}^n$ is the query point that has achieved the best objective value $\bar{\theta}$. We name it the proximal reference point. The proximal analytic center is defined as the solution of

$$\begin{array}{ll} \min\limits_{u,z,\zeta,s_0,s,\sigma} & \Psi(u, s_0, s, \sigma) \\ \text{s.t.} & s_0 + \pi^T z + \zeta = \bar{\theta}, \\ & s_i + (A^T u - E^T z)_i = c_i, \qquad i \in K = \{1, \ldots, \kappa\}, \\ & \sigma + (f_2(u) - \zeta) = 0, \\ & s_0 > 0,\ s > 0,\ \sigma > 0. \end{array} \tag{2.10}$$

If $(u, z, \zeta, s_0, s, \sigma)$ is feasible to (2.10), then (2.10) is equivalent to minimizing $\Phi(u, z, \zeta) = \Psi(u, s_0, s, \sigma)$, in which s_0, s and σ are replaced by their value in u, z and ζ. Note that the localization set is not necessarily compact, but it is easy to show that, thanks to the proximal term, the generalized analytic center exists and is unique.

In the next paragraphs, we shall use the following notation. Given a vector $s > 0$, S is the diagonal matrix whose main diagonal is s. We also use $s^{-1} = S^{-1}e$ to denote the vector whose coordinates are the inverse of the coordinates of s. Similarly, $s^{-2} = S^{-2}e$. Finally, given two vectors x and y of same dimension, xy denotes their component-wise product. With this notation, the first order optimality conditions for (2.10) are

$$\begin{array}{rcll} \rho(u - \bar{u}) + Aws^{-1} + \omega f_2'(u)\sigma^{-1} & = & 0, & (2.11) \\ \pi w_0 s_0^{-1} - Ews^{-1} & = & 0, & (2.12) \\ w_0 s_0^{-1} - \omega\sigma^{-1} & = & 0, & (2.13) \\ s_0 + \pi^T z + \zeta - \bar{\theta} & = & 0, & (2.14) \\ s_i + (A^T u - E^T z)_i - c_i & = & 0, \quad i \in K = \{1, \ldots, \kappa\}, & (2.15) \\ \sigma + f_2(u) - \zeta & = & 0. & (2.16) \end{array}$$

The algorithm that computes the analytic center is essentially a Newton method applied to (2.11)-(2.16). We shall see later how the vector $\xi = ws^{-1}$ is used to derive a lower bound for the optimal solution.

In view of Assumption (1), Φ is self-concordant; Newton's method is thus polynomially convergent [103]. For the sake of simplicity, let us define $v = (u, z, \zeta)$. In the case when v is feasible to (2.10) the Newton direction is

$$dv = -[\Phi''(v)]^{-1}\Phi'(v).$$

The damped Newton's method for computing the proximal analytic center consists in taking damped steps to preserve feasibility of v. The aim is to achieve a sufficient decrease of Φ, until the domain of quadratic convergence is reached. Let

$$\lambda(v) = ([\Phi''(v)]^{-1}\Phi'(v))^T\Phi'(v) = -dv^T\Phi'(v). \tag{2.17}$$

As long as $\lambda(v) > \frac{3-\sqrt{5}}{2}$ a step of length $(1 + \lambda(v))^{-1}$ preserves feasibility and induces a decrease of Φ by an absolute constant. When $\lambda(v) \leq \frac{3-\sqrt{5}}{2}$ a full step is feasible and the method converges quadratically. The method has polynomial complexity.

The stopping criterion is triggered by the proximity measure. When $\lambda(v)$ falls below the threshold value $\eta < \frac{3-\sqrt{5}}{2}$, the search for the proximal analytic center stops. In practice, the much looser criterion $\eta = 0.99$ suffices.

2.6.2 Infeasible Newton's method

Unfortunately we don't have easy access to feasible solution for problem (2.10). In cutting plane schemes, new constraints cut off the current iterate from the new localization set and there is no direct way to retrieve feasibility if the cuts are deep. Since we can't anymore eliminate the variables (s_0, s, σ), we can't apply a feasible Newton method to minimize Φ. Thus, we propose an infeasible start Newton method for (2.10), which aims to achieve feasibility and optimality simultaneously in the extended space $(u, z, \zeta, s_0, s, \sigma)$.

In the course of the optimization process, the first order conditions (2.11)-(2.16) are never satisfied. However, we can assume that $(s_0, s, \sigma) > 0$. We introduce the residuals $r = (r_u, r_z, r_\zeta, r_{s_0}, r_s, r_\sigma)$ and write

$$\begin{aligned}
\rho(u - \bar{u}) + Aws^{-1} + \omega f_2'(u)\sigma^{-1} &= -r_u, && (2.18)\\
w_0\pi s_0^{-1} - Ews^{-1} &= -r_z, && (2.19)\\
w_0 s_0^{-1} - \omega\sigma^{-1} &= -r_\zeta, && (2.20)\\
s_0 + \pi^T z + \zeta - \bar{\theta} &= -r_{s_0}, && (2.21)\\
s_i + (A^T u - E^T z)_i - c_i &= -r_{s_i}, \quad i \in K = \{1, \ldots, \kappa\}, && (2.22)\\
\sigma + f_2(u) - \zeta &= -r_\sigma. && (2.23)
\end{aligned}$$

The Newton direction associated to (2.18)-(2.23) is given by

$$P \begin{pmatrix} du \\ dz \\ d\zeta \\ ds_0 \\ ds \\ d\sigma \end{pmatrix} = \begin{pmatrix} r_u \\ r_z \\ r_\zeta \\ r_{s_0} \\ r_s \\ r_\sigma \end{pmatrix} \tag{2.24}$$

where

$$P = \begin{pmatrix} \rho I + \omega f_2(u)''\sigma^{-1} & 0 & 0 & 0 & -AS^{-2} & \omega f_2(u)'\sigma^{-2} \\ 0 & 0 & 0 & -w_0\pi s_0^{-2} & E^T S^{-2} & 0 \\ 0 & 0 & 0 & -w_0 s_0^{-2} & 0 & \omega\sigma^{-2} \\ 0 & \pi^T & 1 & 1 & 0 & 0 \\ A^T & -E^T & 0 & 0 & I & 0 \\ f_2'(u) & 0 & -1 & 0 & 0 & 1. \end{pmatrix}$$

Since (2.14) and (2.15) are linear, a full Newton step, i.e., a step of length 1, yields a point that is feasible with respect to these equations. However, the same step does not yield a feasible point with respect to the nonlinear equation (2.16). Thus, the method remains essentially infeasible and we cannot use the proximity measure λ to determine the steplength α_{step}. Instead, we use the following empirical rule. Let

$$\alpha_{max} = \max(\alpha \mid s + \alpha ds > 0,\ s_0 + \alpha ds_0 > 0,\ \sigma + \alpha d\sigma > 0),$$

the selected step is

$$\alpha_{step} = \min(1,\ \gamma\alpha_{max}),$$

where the parameter γ is a safeguard to stay away from the boundary of the domain. In practice, we take $\gamma = 0.95$.

When $f_2(u)$ is linear (or constant), it may be the case that (2.14) and (2.15) become satisfied. Instead of using the default step length $(1 + \lambda(v))^{-1}$, as prescribed by the theory, we perform the one-dimensional linesearch

$$\alpha_{step} = \arg\min \Psi(v + \alpha dv).$$

As mentioned earlier, the query point is not feasible for the new cuts returned by the first order oracle. Finding a good starting value for $s_{\kappa+1}$ and/or s_0 after a cut has been added is an issue. Though [66] proposes a scheme that preserves the polynomial complexity of the method, in our practical implementation we use a simple heuristic that turn out to be very efficient.

To summarize, a basic step of the Newton iteration is

1. Send the current point u to the second order oracle to compute the objectif function $f_2(u)$ and its first and second derivatives.

2. Compute the Newton step $(du, dz, d\zeta, ds_0, ds, d\sigma)$ by (2.24).

3. Compute a step length α_{step} to update $(u, z, \zeta, s_0, s, \sigma)$.

4. Test termination.

2.6.3 Lower bound

A lower bound for (2.6) permits a measure of progress to optimality. We now explain a way to generate such a bound. The first step in the derivation of the lower bound consists in introducing the perturbed function $f(u) - r^T u$, where r is a vector to be specified later. The

second step is to replace the non-smooth function $f_1(u)$ by its current polyhedral approximation. This is done by replacing $f_1(u)$ by $\pi^T z$ under the constraints $A^T u - E^T z \leq c$. We thus have the bounding inequality

$$f(u) - r^T u \geq \min_{u,z}\{\pi^T z + f_2(u) - r^T u \mid A^T u - E^T z \leq c\}.$$

In view of the convexity of f_2, we may write

$$\begin{aligned} f(u) - r^T u &\geq f_2(u^c) - f_2'(u^c)^T u^c + \\ &\quad \min_{u,z}\{\pi^T z + f_2'(u^c)u - r^T u \mid A^T u - E^T z \leq c\}, \end{aligned}$$

where u^c is a point of choice (e.g., approximate analytic center). By duality we obtain

$$\begin{aligned} f(u) - r^T u &\geq f_2(u^c) - f_2'(u^c)^T u^c + \\ &\quad \min_{u,z}\max_{\xi\geq 0}\{(f_2'(u^c) + A\xi)^T u + (\pi - E)^T z - c^T\xi - r^T u\}, \\ &= f_2(u^c) - f_2'(u^c)^T u^c + \max_{\xi\geq 0}\Big\{-c^T\xi + \\ &\qquad + \min_{u,z}\left[(f_2'(u^c) + A\xi - r)^T u + (\pi - E\xi)^T z\right]\Big\}. \end{aligned} \tag{2.25}$$

If $\xi \geq 0$ is such that $f_2'(u^c) + A\xi = r$ and $E\xi = \pi$, then

$$f(u) \geq f_2(u^c) - f_2'(u^c)^T u^c + r^T u - c^T\xi.$$

We now show how one can get such a vector ξ at the end of the iterations that compute the proximal analytic center. In view of (2.19), we let $\xi = \xi^c = w(s^c)^{-1} > 0$ and we scale ξ^c by using the special structure of the matrix E to have $\pi - E\xi^c = 0$ and we define $r = f_2'(u^c) + A\xi^c$. In view of the optimality conditions (2.11) and (2.12) one may expect r to be small. We obtain the bound for the optimal objective function value by

$$\begin{aligned} f(u^*) &\geq f_2(u^c) - f_2'(u^c)^T u^c - c^T\xi^c + r^T u^*, \\ &\geq f_2(u^c) - f_2'(u^c)^T u^c - c^T\xi^c + r^T(u^* - u^c) + r^T u^c, \\ &\geq f_2(u^c) - f_2'(u^c)^T u^c + r^T u^c - c^T\xi^c - ||r||\delta. \end{aligned} \tag{2.26}$$

The last inequality follows from Cauchy-Schwartz and $\delta \geq ||u^* - u^c||$ is an upper bound on the distance of the current point u^c to the optimal set. Finding a good value for δ cannot be done on theoretical grounds. It is essentially problem dependent. In practice, we obtained good results by taking the "empirical" value $\delta = 5 \times ||u^c - \bar{u}||$.

If the variable u is constrained to be nonnegative in (2.6), we can further improve the computation of the lower bound by taking $r = -\min\{0, f_2'(u^c) + A\xi^c\}$, where the min operator is taken component-wise. In that case, the coefficient of u in the inner minimization is always nonnegative and $(f_2'(u^c) + A\xi - r)^T u = 0$ at the solution of (2.25). This remark is particularly useful when $r = 0$. Then we obtain the exact lower bound $f_2(u^c) - f_2'(u^c)^T u^c - c^T\xi^c$.

2.6.4 Implementation

Since the oracle is entirely user-defined, we do not include it in the description. The code has two main blocks: the first one computes query points; the second one organizes the dialog between the oracle and the query point generator. The code also includes an important initialization block.

Initialization This module initializes the instance and the various parameters.

Query point generator This modules includes two submodules: the first one creates the localization set based on the information sent by the cut manager; the second one computes approximate proximal analytic centers.

Manager This module keeps track of the cuts generated by the oracle and of the current primal and dual coordinates of the analytic center. It also controls the parameters that are dynamically adjusted and computes criteria values that can be used by the user to stop the algorithm. Finally, it acts as a filter between the oracle and the query point generator.

Two parameters of Proximal-ACCPM are often critical in the applications: the weight w_0 on the epigraph cut in (2.8) and the coefficient ρ of the proximal term in (2.9). The general strategy is to assign to w_0 a value equal to the number of generated cuts [65]. The management of the proximal term is more problem dependent. This point will be briefly commented in the next section. When the problem to be solved has no box constraints on the variables (e.g., when relaxing equality constraints in Lagrangian relaxation) the computation of the Newton direction in Proximal-ACCPM can be made more efficient than in plain ACCPM [51].

The code is written in Matlab; it has around 700 lines of code in the query point generator and 400 in the manager. Matlab is particularly efficient in dealing with linear algebra. Not much gain can be expected by translating the code into C++. However, a C version would make it easier to link Proximal-ACCPM with oracles written in C or FORTRAN or to do an embedding of Proximal-ACCPM within a larger optimization scheme (e.g., a branch and bound scheme). The code is the result of a continuing development efforts by teams at Logilab partly supported by Swiss NSF.

2.7 Applications

We have seen that oracle based optimization is relevant when it is possible to approximate the epigraph set of the function to be minimized, and the feasible set, by polyhedral sets. Let us list a few techniques that lead to this situation: Lagrangian relaxation [64], Lagrangian decomposition [72], column generation [14], Benders' decomposition [18], dual gap function in variational inequalities [101], etc. In this section we present three representative applications, one in large-scale nonlinear continuous optimization, one in combinatorial optimization and one dealing with the coupling of economic and environmental models. Those problems have been fully treated in [8, 6, 16, 33].

In each case, we give a brief presentation of the problem and report a sample of numerical results. This will give the reader an idea of the type of problems that can be solved with Proximal-ACCPM. When the numerical results are displayed in a table, we give the following information: problem identification, denoted 'Problem ID', number of outer iterations (equivalently, the number of oracle calls), denoted 'Outer', number of inner iterations (Newton iterations to compute an analytic center), denoted 'Inner', total CPU time in second, denoted 'CPU' and the fraction of the CPU time spent in the oracle, denoted '%Oracle'.

2.7.1 Multicommodity flow problems

Given a network represented by the directed graph $\mathcal{G}(\mathcal{N}, \mathcal{A})$, with node set $\mathcal{N}$ and arc set $\mathcal{A}$, the multicommodity flow problem consists in shipping some commodity flows from sources to sinks such that the demands for each commodities are satisfied, the arc flow constraints are met and the total cost flow is minimum. The arc-flow formulation of the multicommodity flow problem is

$$\begin{aligned}
\min \quad & \sum_{a \in \mathcal{A}} f_a(y_a) && && (2.27)\\
\text{s.t.} \quad & \sum_{k \in \mathcal{K}} x_a^k = y_a, && \forall a \in \mathcal{A}, && (2.28)\\
& N x^k = d_k \delta^k, && \forall k \in \mathcal{K}, && (2.29)\\
& x_a^k \geq 0, && \forall a \in \mathcal{A},\ \forall k \in \mathcal{K}. && (2.30)
\end{aligned}$$

Here, N is the network matrix; $\mathcal{K}$ is the set of commodities; d_k is the demand for commodity k; and δ^k is vector with only two non-zeros components: a 1 at the supply node and a -1 at the demand node. The variable x_a^k is the flow of commodity k on the arcs a of the network and x^k is the vector of x_a^k. The objective function f is a congestion function on the arcs. For the sake of simpler notation we write problem (2.27) in the more compact formulation

$$\min\{f(y) \mid Bx = y,\ x \in X\}, \tag{2.31}$$

where X represents the set of feasible flows that meet the demands with respect to the network constraints. Bx defines the load flow.

The standard Lagrangian relaxation of (2.31) assigns the dual variables u to the coupling constraints $Bx = y$ and relaxes them. The Lagrangian problem is

$$\max_{u \geq 0} \mathcal{L}(u), \tag{2.32}$$

where

$$\begin{aligned}
\mathcal{L}(u) &= \min_{x \in X, y} f(y) + u^T(Bx - y),\\
&= \min_{y}(f(y) - u^T y) + \min_{x \in X} u^T B x,\\
&= -f_*(u) + \min_{x \in X} u^T B x.
\end{aligned}$$

The function $f_*(u)$ is the Fenchel conjugate of f; it is convex. In the multicommodity case, the second part of the Lagrangian is a sum of $|\mathcal{K}|$ shortest path problems. We denote

$$\mathrm{SP}(\bar{u}) = \min_{x \in X}(B^T \bar{u})^T x. \tag{2.33}$$

We recall that in Proximal-ACCPM, we treat the negative of the objective function (2.32). Let $\bar{x}$ be an optimal solution returned by the oracle (2.33) at a given point $\bar{u}$. Since $\mathrm{SP}(u)$ results from a minimization problem, the inequality $\mathrm{SP}(u) \leq (B\bar{x})^T u$ provides a linear upper estimate of the concave function $\mathrm{SP}(u)$. The solution computed by the oracle $-f_*(\bar{u}) + (B\bar{x})^T \bar{u}$ produces a lower bound for the original problems. Instead of using (2.26) to compute an upper bound, we use the variable ξ to compute a feasible solution to (2.27) (It can be shown).

For the nonlinear multicommodity flow problem, we use the most widely used function in telecommunications, the so-called Kleinrock congestion function:

$$f(y) = \frac{y}{c - y},$$

where c is the vector of capacities on the arcs. The conjugate function is

$$f_*(u) = 2\sqrt{c^T u} - c^T u - 1, \quad \forall u \geq \frac{1}{c}.$$

For the linear case, the objective function is

$$f(y) = \begin{cases} t^T y, & 0 \leq y \leq c, \\ +\infty, & \text{otherwise}, \end{cases}$$

where c is the vector of capacities and t the vector of unit shipping cost on the arcs. The conjugate function is

$$f_*(u) = c^T u, \quad \forall u \geq 0.$$

To get a feel for the numerical performance, we pick few examples that have been solved in [8, 6]. We select 3 types of problems. `Planar` and `Grid` instances are telecommunications networks while `Winnipeg`, `Barcelona` and `Chicago` are transportation problems. Table 2.1 gives for each problem the number of nodes, the number of arcs, and the number of commodities. The oracle is a shortest path problem solved with Dijkstra algorithm. The code is written in C. The tests were performed on a PC (Pentium IV, 2.8 GHz, 2 Gb of RAM) under Linux operating system.

Table 2.2 shows the numerical results to solve the linear and the nonlinear case with a relative otpimality gap less than 10^{-5}. We followed different strategies in the management of the proximal term, depending on whether the problem is linear or not. In the linear case, a constant value for the proximal parameter, say $\rho = 10^{-2}$ is suitable. In the nonlinear case, the proximal parameter is dynamically adjusted, according to success or failure in improving the value of the Lagrangian dual objective (lower bound). We start with $\rho = 1$ and multiply the current ρ by 10 in case of a 3 consecutive failures, up to the limit value $\rho = 10^{10}$.

Problem ID	# nodes	# arcs	# commodities
`planar500`	500	2842	3525
`planar800`	800	4388	12756
`planar1000`	1000	5200	20026
`grid12`	900	3480	6000
`grid13`	900	3480	12000
`grid14`	1225	4760	16000
`grid15`	1225	4760	32000
`Winnipeg`	1067	2975	4345
`Barcelona`	1020	2522	7922
`Chicago`	933	2950	93513

Table 2.1: Test problems.

	Linear case				Nonlinear case			
Problem ID	Outer	Inner	CPU	%Oracle	Outer	Inner	CPU	%Oracle
`planar500`	229	744	88.7	21	127	324	32.2	37
`planar800`	415	1182	557.2	16	182	429	110.5	40
`planar1000`	1303	2817	7846.7	12	381	869	568.1	26
`grid12`	509	1341	658.5	18	201	409	106.7	41
`grid13`	673	1629	1226.8	12	222	454	128.7	39
`grid14`	462	1363	843.6	22	204	414	173.2	48
`grid15`	520	1450	1055.1	20	203	414	172.8	48
`Winnipeg`	224	592	81.2	18	338	988	215.0	14
`Barcelona`	157	421	35.9	23	253	678	101.1	15
`Chicago`	180	493	79.2	47	145	370	48.6	41

Table 2.2: Numerical results.

The results in Table 2.2 have been further improved by means of column elimination and an active set strategy. With these enhancements, the method could solve huge instances with up to 40,000 arcs and 2,000,000 commodities. It has also been compared to other state-of-the-art methods. It appears to be very competitive, especially in the linear case, where it turns out to be from 4 to 30 times faster than the best known results. (For more details, see [8, 6].)

Let us also mention that the impact of the proximal term has been analyzed to some depth in the two papers cited above. The introduction of a proximal term in ACCPM instead of box constraints on the variables has proved to be beneficial in almost all cases. It never appeared to be detrimental. On nonlinear multicommodity flow problems or on linear problems with an advanced treatment (column elimination, active set strategy) the version with the proximal term outperformed the version with box constraints.

2.7.2 Lagrangian relaxations of the p-median problem

In the p-median problem the objective is to open p 'facilities' from a set of m candidate facilities relative to a set of n 'customers', and to assign each customer to a single facility. The cost of an assignment is the sum of the shortest distances c_{ij} from a customer to a facility. The distance is sometimes weighted by an appropriate factor, e.g., the demand at a customer node. The objective is to minimize this sum. Applications of the p-median problem can be found in cluster analysis, facility location, optimal diversity management problem, etc. [27]. The p-median problem is NP-hard [83].

The p-median problem can be formulated as follows

$$\min_{x,y} \quad \sum_{i=1}^{m}\sum_{j=1}^{n} c_{ij}x_{ij} \tag{2.34}$$

$$\text{s.t.} \quad \sum_{i=1}^{m} x_{ij} = 1, \quad \forall j, \tag{2.35}$$

$$\sum_{i=1}^{m} y_i = p, \tag{2.36}$$

$$x_{ij} \leq y_i, \quad \forall i, j, \tag{2.37}$$

$$x_{ij}, y_i \in \{0, 1\}, \tag{2.38}$$

where $x_{ij} = 1$ if facility i serves the customer j, otherwise $x_{ij} = 0$ and $y_i = 1$ if we open facility i, otherwise $y_i = 0$.

In the following two sections we formulate the (standard) Lagrangian relaxation of the p-median problem, and the semi-Lagrangian relaxation.

Standard Lagrangian relaxation of the p-median problem

In this section we focus in the resolution of the (standard) Lagrangian relaxation (LR) of the p-median problems by means of Proximal-ACCPM. To this end, we relax constraints (2.35) and (2.36) in (2.34), to yield the dual problem

$$\max_{u,v} \mathcal{L}_1(u, v), \tag{2.39}$$

and the oracle

$$\mathcal{L}_1(u, v) = \min_{x,y} \quad \sum_{i=1}^{m}\sum_{j=1}^{n} c_{ij}x_{ij} + \sum_{j=1}^{n} u_j(1 - \sum_{i=1}^{m} x_{ij}) + v(p - \sum_{i=1}^{m} y_i) \tag{2.40}$$

$$\text{s.t.} \quad x_{ij} \leq y_i, \quad \forall i, j, \tag{2.41}$$

$$x_{ij}, y_i \in \{0, 1\}, \tag{2.42}$$

where $u \in \mathbb{R}^n$ is associated to the constraints $\sum_{i=1}^{m} x_{ij} = 1, \quad j = 1, \ldots, n$, and $v \in \mathbb{R}$ to the constraint $\sum_{i=1}^{m} y_i = p$.

We name *Oracle 1* this oracle; it is trivially solvable. Its optimal solution is also optimal for its linear relaxation. Consequently, the optimum of $\mathcal{L}_1$ coincides with the optimum of the linear relaxation of (2.34).

To show Proximal-ACCPM performance when solving the standard Lagrangian relaxation (2.40), we take a few examples reported in [51]. In this technical report, several p-median problems based on data from the *traveling salesman problem* (TSP) library [110] are solved. Instances of the grid problem, where the customers are regularly spaced points on square, are also solved. In Table 2.3 we show the results for ten representative instances (Proximal-ACCPM stopping criterion set equal to 10^{-6}). In this case, the proximal parameter is set to $\rho = 1$ initially and is dynamically adjusted by multiplicative factors 2 and 0.5 depending on the success or failure in improving the objective of the Lagrangian dual objective. The updating is limited by the bounds 10^{-6} and 10^4. Programs have been written in MATLAB

and run in a PC (Pentium-III PC, 800 MHz, with 256 Mb of RAM) under the Linux operating system.

Problem ID	n	p	Outer	Inner	CPU	%Oracle
Grid1521	1521	10	348	902	132	33
Grid1849	1849	10	417	1042	241	32
Grid2025	2025	10	382	961	229	37
Grid2304	2304	10	448	1111	370	34
Grid2500	2500	10	440	1095	428	34
TSP1817	1817	10	1070	2303	1861	10
TSP2103	2103	10	316	701	156	48
TSP2152	2152	10	196	430	98	51
TSP2319	2319	10	369	775	237	46
TSP3038	3038	10	127	292	102	62

Table 2.3: Numerical results.

Semi-Lagrangian relaxation of the p-median problem

The standard Lagrangian relaxation is commonly used in combinatorial optimization to generate lower bounds for a minimization problem. An optimal integer solution is obtained by a branch and bound scheme. The semi-Lagrangian relaxation (SLR) is a more powerful scheme, introduced in [16], that generates an optimal integer solution for (linear) combinatorial problems with equality constraints.

To strengthen $\mathcal{L}_1$, the SLR introduces in (2.34) the redundant constraints $\sum_i x_{ij} \leq 1$, $j = 1, \ldots, n$, and $\sum_i y_i \leq p$. After relaxing (2.35-2.36), we obtain the SLR dual problem

$$\max \mathcal{L}_3(u, v), \tag{2.43}$$

and the new oracle

$$\begin{aligned}
\mathcal{L}_3(u,v) \quad = \min_{x,y} \quad & \sum_{i=1}^{m}\sum_{j=1}^{n} c_{ij}x_{ij} + \sum_{j=1}^{n} u_j(1 - \sum_{i=1}^{m} x_{ij}) + v(p - \sum_{i=1}^{m} y_i) && (2.44)\\
\text{s.t.} \quad & \sum_{i=1}^{m} x_{ij} \leq 1, \quad \forall j, && (2.45)\\
& \sum_{i=1}^{m} y_i \leq p, && (2.46)\\
& x_{ij} \leq y_i, \quad \forall i, j, && (2.47)\\
& x_{ij}, y_i \in \{0, 1\}. && (2.48)
\end{aligned}$$

This oracle, which we name *Oracle 3*, is much more difficult than Oracle 1 (in fact, Oracle3 is NP-hard). To cope with this difficulty one can use an intermediate oracle (*Oracle 2*) defined as the Oracle 3 but without constraint (2.46). We denote $\mathcal{L}_2$ the associated dual function. In general, Oracle 2 is easier to solve than Oracle 3, especially in cases where the p-median underlying graph associated to Oracle 2 decomposes into independent subgraphs. In such situation, we solve an integer problem per subgraph (see [16] for more details).

It can be seen that solving the SLR dual problem (2.44) completely solves the p-median problem. Based on this result, we design a branch-and-bound free procedure to completely solve the p-median problem. This procedure successively maximizes the dual functions $\mathcal{L}_i(u,v), i = 1,2,3$. In this succession of three dual problems, the optimal solution of one dual problem is used as the starting point for the next dual problem. After solving the last dual problem ($\mathcal{L}_3(u,v)$) we obtain, as a by-product, an optimal integer solution for (2.34). These dual problems are solved by means of Proximal-ACCPM. Oracle 2 and 3 are solved by means of CPLEX 8.1. Note that, although our procedure is branch-and-bound free, CPLEX is, of course, based on a sophisticated branch-and-bound procedure.

If we are not able to solve the three dual problems we will only have a lower bound of the p-median optimal value. In this case, we will compute an integer solution for the p-median problem by means of an heuristic as for example the 'Variable Neighborhood Decomposition Search' (VNDS) [76]. The quality or the integer solution will be determined by the dual lower bound.

In Tables 2.4 and 2.5 we show the results (solution quality and performance) for 10 representative examples of the 44 instances tested in [16]. These instances can be found in the TSPLIB [110] and range from 1304 to 3795 customers, which implies 2 to 14 million binary variables. The proximal parameter is set to the constant value $\rho = 10^{-2}$ for problems with Oracle 2 and Oracle 3. In these tables 'Or.' stands for Oracle, 'VNDS' for variable neighborhood decomposition search, 'SLR' for semi-Lagrangian relaxation and 'ANIS' for averaged number of independent subgraphs. '%Opt.' gives the quality of the solution and is computed as

$$100 \times \left(1 - \frac{\text{'Upper bound'} - \text{'Lower bound'}}{\text{'Lower bound'}}\right).$$

Programs have been written in MATLAB and run on a PC (Pentium-IV Xeon PC, 2.4 GHz, with 6 Gb of RAM) under the Linux operating system. Note that in some cases the Oracle 3 is not called. The reason is either because the problem has been completely solved by the second dual problem or the CPU time limit has been reached when solving the second dual problem.

Instance			Lower bound			Upper bound		%Opt.
Problem ID	n	p	Or. 1	Or. 2	Or. 3	Value	Method	
rl1304	1304	10	2131787.5	2133534	-	2134295	VNDS	99.96
rl1304	1304	500	97008.9	97024	-	97024	SLR	100
vm1748	1748	10	2982731.0	2983645	-	2983645	SLR	100
vm1748	1748	500	176976.2	176986	176986	176986	SLR	100
d2103	2103	10	687263.3	687321	-	687321	SLR	100
d2103	2103	500	63938.4	64006	64006	64006	SLR	100
pcb3038	3038	5	1777657.0	1777677	-	1777835	VNDS	99.99
pcb3038	3038	500	134771.8	134798	134798	136179	VNDS	98.98
fl3795	3795	150	65837.6	65868	-	65868	SLR	100
fl3795	3795	500	25972.0	25976	25976	25976	SPR	100

Table 2.4: Solution quality

Instance			Outer			ANIS	CPU			
Problem ID	n	p	Or. 1	Or. 2	Or. 3		Or. 1	Or. 2	Or. 3	Total
rl1304	1304	10	390	35	0	1	95	17241	0	17336
rl1304	1304	500	133	15	0	143	8	40	0	48
vm1748	1748	10	500	21	0	1	174	3771	0	3945
vm1748	1748	500	146	15	2	131	14	61	22	97
d2103	2103	10	241	7	0	2	41	504	0	545
d2103	2103	500	500	26	2	39	143	10086	4309	14538
pcb3038	3038	5	341	5	0	1	111	1988	0	2099
pcb3038	3038	500	211	17	2	38	56	3269	3900	7225
fl3795	3795	150	1000	27	0	17	1100	39199	0	40299
fl3795	3795	500	500	38	1	25	259	2531	218	3008

Table 2.5: Performance

2.7.3 Coupling economic and environmental models

Integrated assessment of environmental (IAM) policies is becoming an important priority due to the social need for local air pollution control or global climate change mitigation. Typically an IAM will combine an economic model and an environmental model to yield an evaluation of the costs and benefits associated with some environmental goals, given the technological and economic choices that are available. In this section we present a successful implementation using Proximal-ACCPM in this context.

In [77], it has been proposed to use an oracle-based method to couple an Eulerian air quality model and a techno-economic model of energy choices in an urban region. The implementation of the approach has been further developed and tested in [33]. Ozone (O_3) pollution is usually modelled in so-called Eulerian models that represent the transport of primary pollutants (typically NO_x and VOCs) and the air photochemistry under various weather conditions and for the specific topography of the region considered. These models take the form of large scale distributed parameter systems that are run over specific "weather episodes" (for example a two-day summer sunny period which may amplify the probability of ozone peaks in green areas). These simulations serve to build air-quality indicators like, e.g. the *ozone concentration peak* or *the average over a threshold* (AOT) during an episode. On the other side techno-economic models are dynamic capacity expansion and production models, also called activity analysis models. A typical example is MARKAL, initially developed to represent energy-technology choices at a country level (see [58], [20]) and also adapted to the description of these choices at a city level in [60] and [59]. In a MARKAL model the planning horizon is in general defined as 9 periods of 5 years. The model finds, for specified demands in energy services, world prices of imported energy and given a gamut of technology choices, an investment plan and a production program that minimize a system-wide total discounted cost while satisfying some pollutant emissions limits.

From this brief description of the two categories of models, the reader may realize that they belong to very different worlds. The interaction of the models in a coupling procedure can be schematized as follows. The economic model produces a vector of pollutants emissions per sector of activity. These emissions are then distributed over time and space using patterns that depend on the type of activity. For instance, global urban heating emissions are easily dispatched in space using the geographical distribution of buildings. They are also distributed in time to follow a yearly seasonal pattern. The other important cause of emission is the volume of traffic. The economic activity analysis proposes a list of technologies used in different transport sectors (cars, public transport, taxis, etc), resulting in a global

emission level for each of these sectors. To obtain the spatio-temporal distribution of these emissions due to traffic one resorts to a complex congestion model of traffic, that essentially computes traffic equilibria. These different sources of pollutant emissions are then combined into a spatio-temporal distribution map of emissions. The last step in the analysis consists in simulations performed with the Eulerian model to compute air quality indices on a set of critical episodes. The combination of models that eventually produces the air quality indices is complex, but at the end one can effectively compute air quality indices as a function of the global emissions of pollutants by sector of economic activity. Clearly, one cannot expect this function to be linear. Even worse, the computation may be very time consuming.

We have described a one-way interaction of the models, starting from the economic model and ending with air quality indices. Let us now describe the feedback from the air quality assessment. Indeed, one may want to limit peaks of pollution. This can be translated into upper limits on the air quality indices. We now study this reverse mechanism and show how the complete problem can be recast in the format of problem (2.6). Let us first schematize the economic activity analysis as the linear program

$$\min\{c^T x \mid Ax = a,\ x \geq 0\}. \tag{2.49}$$

We shall refer to it as the E^3 model. The economic activity x induces a vector y of pollutants emissions. This vector is indexed by sector of activity. In the paradigm of linear activity analysis, the total emission vector is assumed to be a linear function of the economic activity level, say

$$y = Bx.$$

The complex transformation of the vector y of sectorial emissions into air quality indices is represented by a vector function $\Pi(y)$. In [33] it is shown that one can compute the function value and estimate its gradient at any point y. If $\bar{\Pi}$ is the bound imposed on the air quality indices (higher indices imply lower air quality), we can represent our complex problem as the mathematical programming problem

$$\min\{c^T x \mid Ax = a,\ Bx - y = 0,\ \Pi(y) \leq \bar{\Pi},\ x \geq 0\}. \tag{2.50}$$

This large-scale highly nonlinear model is intractable by standard optimization tool. However, it is quite easily amenable to an Oracle Based Optimization approach. To this end, we introduce the function

$$f(y) = \min\{c^T x \mid Ax = a,\ Bx = y,\ x \geq 0\}, \tag{2.51}$$

and the set

$$Y = \{y \mid \Pi(y) \leq \bar{\Pi}\}. \tag{2.52}$$

Our original problem can now be written as

$$\min\{f(y) \mid y \in Y\}.$$

It remains to show that the above problem is of the same type as (2.6). It is a well-known fact of convex analysis that the function $f(y)$ is convex (this is easily seen by considering the dual of the linear program that defines f) and that one can compute a subgradient at each point of the domain of the function. Unfortunately, one cannot make a similar statement on Y. Being the result of such a complex transformation process, $\Pi(y)$ is likely to be nonconvex. However,

one can hope that in the range of values that are of interest the nonconvexity is mild. This is supported by empirical evidence. A gradient is also estimated by a finite difference scheme.

Even in presence of mild nonconvexity, one cannot exclude pathology in running Proximal-ACCPM. A separating hyperplane for the set Y may turn out to cut off part of the set, and exclude a point that was proved to be feasible earlier. To cope with this difficulty, the authors of [33] simply shifted the plane to maintain feasibility. They also made problem (2.51) easier by assuming monotonicity that made it possible to replace the equality constraint $Bx = y$ by $Bx \leq y$.

As the air chemistry description actually involves nonlinear functions, we have implemented a technique of successive local linearization of the air pollution dynamic equations. The details of the implementation are given in [33]. In a particular simulation based on data describing the Geneva (Switzerland) region, a solution to the reduced order optimization problem is obtained through Proximal-ACCPM, with 30 calls to the oracles (24 feasibility cuts and 6 optimality cuts were peformed). A feasibility cut (call to the air quality oracle) takes 30 minutes computing time (SUN Ultra-80, Ultrasparc driver) whereas an optimality cut (call to the techno-economic model) takes 10 seconds.

This application demonstrates the possibilities offered by an OBO method to tackle Integrated Assessment Models where part of the modeling is a large-scale simulator of complex physics and chemistry processes. Since Proximal-ACCPM keeps the number of oracle calls to a small or moderate size it permits the use of these simulators in the design of some oracles and therefore it realizes the coupling that is the essence of IAMs.

Remark A similar implementation has been realized recently for an IAM of climate change policies. It is reported in [47, 48]. In that case the coupling is realized between an economic growth model and an intermediate complexity climate model. This second successful experience that we will not further described here confirms the potential of OBO techniques for the exploitation of complex and large-scale IAMs.

2.7.4 Linear pattern separation

This section summarize our contribution to the *linear separation problem* using PROX-ACCPM[118]. Linear separation [24, 92] is an important concept in data mining [75]. It is widely used and has been applied in many fields, e.g., *cancer diagnosis* [93], *human genome* [78], *game strategies* [85], *pattern recognition* [94], *decision/selection making* [126], and others. Many other separation rules can be found in the literature, and our method can handle those of them that are based on a functional rule expressed by a convex formulation. Since the qualitative aspect of these rules is not the main goal of this work, we consider the linear separation rule as an illustration of our approach. The problem of finding a satisfactory linear separation can be formulated as a mathematical programming problem. In some cases, the size of the data set [55] is so large that solving the mathematical programming problem becomes a challenge even with the state-of-the-art optimization software. We propose to consider an alternative NDO formulation of the problem and use the cutting plane methodACCPM to find the optimum of the objective function efficiently.

The purpose of the *linear separation* is to find a linear function to separate a given set of multi-attribute items that are partitioned into two subsets. In general it is unlikely that a perfect separation exists. Thus, one has to look for an approximative separation with the minimum error. A natural way is to find a separation plane that minimizes the total number of misclassified instances. Unfortunately this leads to a mixed integer programming

problem, which may be very hard even for moderate size data. A more tractable approach [93] consists of minimizing the total deviation (or gap) of the misclassified instances. This approach can be handled through a pure linear programing formulation, or a convex non-differentiable formulation. We consider the second approach using analytic center cutting planes solver[70, 7] and compare it with the LP approach.

Given a set of points $\mathcal{A} = \{a_i \in \mathcal{R}^n, i = 1, 2, \cdots, N\}$, and a partition $\mathcal{S}_1 \cup \mathcal{S}_2$ of the set of indices $\mathcal{S} = \{1, 2, \cdots, N\}$, we wish to find $w \in \mathcal{R}^n$ and $\gamma \in \mathcal{R}$ such that the hyperplane $\{x \mid w^T x = \gamma\}$ separates the two subsets $\mathcal{A}(\mathcal{S}_1)$ and $\mathcal{A}(\mathcal{S}_2)$, where

$$\mathcal{A}(\mathcal{S}_1) = \{a_i \in \mathcal{A} \mid i \in \mathcal{S}_1\}, \tag{2.53}$$
$$\mathcal{A}(\mathcal{S}_2) = \{a_i \in \mathcal{A} \mid i \in \mathcal{S}_2\}. \tag{2.54}$$

For typographical convenience, we will write (w, γ) instead of (w^T, γ).

Actually, one looks for a strong separation. Thus, given a *separation margin* $\nu > 0$, we hope to achieve the separation properties (2.55-2.56) displayed bellow

$$\forall a_i \in \mathcal{A}(\mathcal{S}_1) \quad w^T a_i \geq \gamma + \nu, \tag{2.55}$$
$$\forall a_i \in \mathcal{A}(\mathcal{S}_2) \quad w^T a_i \leq \gamma - \nu. \tag{2.56}$$

In general, there is no guarantee that the two sets can be strongly separated. Therefore, for any choice of w and γ, we might observe *misclassification errors*, which we define as follows

$$e_i^1 = \frac{\max(-w^T a_i + \gamma + \nu, 0)}{||(w, \gamma, \nu)||}, \quad i \in \mathcal{S}_1, \tag{2.57}$$
$$e_i^2 = \frac{\max(w^T a_i - \gamma + \nu, 0)}{||(w, \gamma, \nu)||}, \quad i \in \mathcal{S}_2. \tag{2.58}$$

Our goal is then to build a separation hyperplane $\{x \mid w^T x = \gamma\}$ (i.e., compute w and γ) for which the total sum of *misclassification errors* is minimal. In other words, we want to find a vector w and a scalar γ such that the average sum of misclassifications errors is minimized [92].

The separation margin ν helps avoiding the useless trivial solution $(w, \gamma) = (0, 0)$. Its value is usually set to 1. In some cases the separation margin may lead to large values for w and γ. It may be necessary [55] to bound w to avoid this undesirable feature; so, we add the constraint $||w||_2 \leq k$.

Formally, we have to solve the following optimization problem

$$\min_{(w,\gamma) \in \mathcal{R}^n \times \mathcal{R}} \quad \left[\frac{1}{|\mathcal{S}_1|} \sum_{i \in \mathcal{S}_1} \max(-w^T a_i + \gamma + \nu, 0) + \frac{1}{|\mathcal{S}_2|} \sum_{i \in \mathcal{S}_2} \max(w^T a_i - \gamma + \nu, 0) \right] \tag{2.59}$$

$$\text{subject to:} \quad ||w||_2 \leq k. \tag{2.60}$$

The objective function is the sum of two functions $f = f_1 + f_2$, f_1 and f_2 being themselves

the sum of elementary functions

$$f_1(w,\gamma) = \frac{1}{|\mathcal{S}_1|}\sum_{i\in\mathcal{S}_1}\max(-w^T a_i+\gamma+\nu,0), \tag{2.61}$$

$$f_2(w,\gamma) = \frac{1}{|\mathcal{S}_2|}\sum_{i\in\mathcal{S}_2}\max(w^T a_i-\gamma+\nu,0). \tag{2.62}$$

The following vectors

$$\xi_1 = \frac{1}{|\mathcal{S}_1|}\sum_{i\in\mathcal{S}_1|-w^T a_i+\gamma+\nu>0}(-a_i,1)(w,\gamma) \tag{2.63}$$

$$\xi_2 = \frac{1}{|\mathcal{S}_2|}\sum_{i\in\mathcal{S}_2|w^T a_i-\gamma+\nu>0}(a_i,-1)(w,\gamma) \tag{2.64}$$

are subgradients $\xi_1\in\partial(f_1)$ and $\xi_2\in\partial(f_2)$ of the two functions of interest.

In ACCPM the square normed in the constraint $||w||^2\le k^2$ is also treated as black box. If $\bar{w}$ is not feasible ($||\bar{w}||^2>k^2$), the constraint

$$w+2\langle\bar{,}w-\bar{w}\rangle\le k^2 \tag{2.65}$$

holds for any feasible point.

Finally, let us give two bounds on f. Since $f(0,0)=2\nu$, then 2ν is an upper bound of the optimal value of the objective. A straightforward lower bound is 0, but this can be only attained if perfect classification is achieved.

Let us discuss now the formulation of problem (2.59)–(2.60) as a standard mathematical programming problem. Let z_i, $i\in\mathcal{S}$. be an auxiliary variable. The original problem becomes

$$\min_{\substack{(w,\gamma)\in\mathcal{R}^n\times\mathcal{R}\\ z\ge 0}} \quad \frac{1}{|\mathcal{S}_1|}\sum_{i\in\mathcal{S}_1}z_i+\frac{1}{|\mathcal{S}_2|}\sum_{i\in\mathcal{S}_2}z_i \tag{2.66}$$

$$\text{subject to:} \quad z_i\ge(-w^T a_i+\gamma+\nu),\ i\in\mathcal{S}_1 \tag{2.67}$$

$$z_i\ge(w^T a_i-\gamma+\nu),\ i\in\mathcal{S}_2 \tag{2.68}$$

$$||w||_2\le k. \tag{2.69}$$

Note that the constraints (2.67)–(2.68) are numerous but linear. The problem is thus a large linear programming problem with one quadratic constraint (2.69). Some authors [55] prefer to replace the quadratic constraint by a quadratic penalty term in the objective. Another possibility consists in replacing the Euclidean norm in (2.69) by the ℓ_∞ norm [94], thus obtaining a fully linear formulation.

We compare our approach with direct methods based on a pure linear programming formulation of the problem (i.e. equations (2.66)–(2.68)). The norm constraint is dropped. The dual problem can be formulated as follows:

$$\begin{aligned}
\max \quad & \nu\sum_{i\in\mathcal{S}}\xi_i\\
\text{s.t.} \quad & H\xi=0\\
& 0\le\xi_i\le\frac{1}{|\mathcal{S}_1|},\ i\in\mathcal{S}_1\\
& 0\le\xi_i\le\frac{1}{|\mathcal{S}_2|},\ i\in\mathcal{S}_2.
\end{aligned}$$

Here $\xi \in \mathcal{R}^{|\mathcal{S}|\times 1}$ and

$$H = \begin{bmatrix} A(\mathcal{S}_1) & -A(\mathcal{S}_2) \\ -e(\mathcal{S}_1) & e(\mathcal{S}_2) \end{bmatrix}. \tag{2.70}$$

e is a row vector of 1 of appropriate dimension.

The two equivalent formulations provided above can be solved using standard techniques of linear programming such as *simplex* or *interior point methods*. We have compared ACCPM with two linear programming codes: MOSEK[91] and CPLEX[32]. Both offer the options between a simplex and a primal-dual log barrier algorithm. Table 2.6 displays the timings (in seconds) we have obtained.

n	m	MOSEK(1)	CPLEX(1)	MOSEK(2)	CPLEX(2)	ACCPM
10	10000	2.54	1.95	6.31	1.81	1.95
20	10000	2.21	2.47	11.37	2.35	2.63
30	10000	4.45	4.45	18.70	4.12	3.40
40	10000	6.34	6.11	23.78	5.99	4.50
50	10000	9.23	8.18	26.11	8.20	4.95
60	10000	11.84	11.10	30.78	11.10	6.30
70	10000	15.13	13.07	40.87	12.82	8.86
80	10000	19.87	15.00	50.21	14.21	10.16
90	10000	26.04	19.97	69.20	19.46	15.03
100	10000	30.22	22.19	62.63	21.29	16.81
10	100000	143.86	81.08	113.40	78.12	5.11
20	100000	120.47	109.53	132.25	108.29	8.22
30	100000	172.21	143.98	179.24	141.31	10.59
40	100000	253.38	194.34	215.89	190.40	16.31
50	100000	311.35	223.71	280.87	219.60	16.95
60	100000	576.77	273.46	303.09	285.38	18.97
70	100000	742.84	408.01	411.66	396.50	28.88
80	100000	850.15	427.15	478.61	406.19	31.25
90	100000	906.57	496.57	590.95	439.29	34.06
100	100000	1443.25	543.25	680.81	493.78	40.30

(1) simplex
(2) interior point

Table 2.6: Comparison between LP and NDO aproaches

2.7.5 Cardinality bounded portfolio selection

The classical approach to study the portfolio selection problem[38] is based on the Markowitz mean-variance formulation [97, 96]. The mean (resp. the variance) of a portfolio configuration represents the benefit (resp. the risk) associated with the corresponding investment. The approach boils down to a simple quadratic convex programming problem that is easily solved by nowadays standards [32, 91]. A number of authors have studied this problem in different aspects [74, 90, 99, 111, 131, 132].

The problem can be considered through different aspects: an upper bound on the potential investment, a limit in the number of assets that can be selected (*cardinality constraint*), a lower(resp. upper) bound on the risk(resp. reward) for each selected asset. Other kind of meaningful constraints have been considered in the literature. Jobst *et al* [80] have studied the case of a portfolio with a *fixed* number of assets. We consider an upper bound instead. For investors, the cardinality constraint is important for monitoring and control purposes. Chang *et al* [34] have proposed some heuristics (genetic algorithm, tabu search, and simulated annealing) for the cardinality constraint. There is also the so-called *buy-in-threshold*, which specifies the minimum investment level, and therefore eliminates small trades in the selection. We also consider *buy-in-limit*, which is the maximum investment level (see Bienstock [21] and Lee-Mitchell[86]). It is obvious that bounding the investments has an impact on the number of selected assets. Another interesting constraint is the *roundlots*, which are discrete numbers

of assets taken as the basic unit of the selection. The size of the portfolio is an integer linear combination of the roundlots. The feasibility problem coming from the roundlots constraints has been established to be NP-complete [95].

We extend the standard mean-variance portfolio model by considering three special constraints. First, we assume a fixed transaction cost on each item. Transaction costs are also considered in [1, 129]. Second, we impose a bound on the total number of items in the portfolio. Third, we consider lower and upper limits on the amount that can be invested on each item. The resulting mathematical programming problem is a quadratic mixed integer program. The restriction on the investment levels yields an additional feasibility constraint, which makes the problem more difficult. For a given distribution of the investments, selecting the appropriate set of asset is similar to the *bounded knapsack problem*. We have provided a heuristic that performs well on common instances. Our heuristic is used on the nodes of the branch and bound process (which we have implemented) to obtain upper bounds and hopefully prune some nodes accordingly. The node problem, which a quadratic nondifferentiable optimization instance, is solved using an analytic center cutting planes solver[70, 7].

In our contribution [117], we consider a more general formulation of the problem from the mathematical point of view. Let $V \in R^{n\times n}$ be the variance-covariance matrix of the n available assets, and $r \in R^n$ the vector of expected returns. We consider the expected returns from an individual basis (at the level of the asset), while usual formulations consider a global expectation[34]. The decision variable $x \in R^n$ represents the fractional shares for each of the assets in the portfolio. By definition, x belongs to the simplex, that is $e^T x = 1$, where $e = (1, \cdots, 1)^T$, and $x \geq 0$. The lower and upper bounds on the investment are given by the positive vectors $d, f \in R^n$, $0 \leq d, f \leq 1$. We shall denote D (resp. F) the diagonal matrix with main diagonal d (resp. f). Next, we define the vector of transaction costs $h \in R^n$, and $p \leq n$ is the maximum number of assets that can be selected. Finally, we consider a selection variable $y \in \{0,1\}^n$ ($y_i = 1$ if asset i is selected, and 0 otherwise). We formulate the portfolio selection problem as follows:

$$\min_{x,y} \quad \frac{1}{2}x^T V x - \mu r^T x + h^T y \tag{2.71a}$$

$$e^T x = 1, \tag{2.71b}$$

$$Dy \leq x \leq Fy, \tag{2.71c}$$

$$e^T y \leq p, \tag{2.71d}$$

$$x \in [0,1]^n, \tag{2.71e}$$

$$y \in \{0,1\}^n. \tag{2.71f}$$

As previously mentioned, the literature commonly considers a global expectation of the returns, which means a constraint of the form $r^T x = R_{expected}$. In our formulation, we have included this into the objective to be minimized through the term $-\mu r^T x$, where μ is a parameter (the unit of the return). Note that, for a given asset i, $x_i = 0$ (null proportion) is equivalent to $y_i = 0$ (not selected). This is well represented by the bound constraint $d_i y_i \leq x_i \leq f_i y_i$, written in a matrix form (2.71c). Our purpose is to provide an efficient framework to solve the portfolio selection selection problem based on the formulation (2.71), based on the *cutting planes method* (with ACCPM), *Bender decomposition*, and the *branch and bound*.

We now report our computational results obtained with a MATLAB implementation of our algorithms on a 2.5 Ghz processor with 2 Go of memory. The quadratic programming

subproblem was solved using MOSEK [91]. The first group of data has been generated randomly using (the `rand` routine of MATLAB), which follows a normal distribution. The second group is a collection of five data sets drawn from the Hang Seng, Dax, FSTE, S&P, and Nikkei indices [34].

n	p	nodes	time(s)
30	10	37	267
30	15	14	122
30	20	22	226
50	5	25	155
50	10	160	878
100	10	13	95
100	20	527	2734
200	10	8	167
200	20	715	5639
300	10	74	3200
300	20	287	6300
400	10	238	17240
400	30	33	551
500	10	181	20016
500	50	429	8922

Table 2.7: Global performance on randomly generated data sets

Set	n	p	nodes	time(s)
Hang Seng	31	10	28	140.0
DAX	85	10	3	14.5
FTSE	89	10	3	28.0
S&P	98	10	5	40.8
Nikkei	225	10	92	2837.5

Table 2.8: Global performance on Beasley collection

As we can see in Table 2.7 and Table 2.8, our algorithm solves the portfolio selection problem quite efficiently. The results are better with lower values of p (the maximum number of selections). It is clear that we could have obtained better timings if we were using a compiled binary version of our codes.

2.8 Conclusion and perspectives

We have presented Proximal-ACCPM, an efficient method for convex nondifferentiable optimization, and discussed five large-scale applications that are representative of an oracle based optimization approach. The last application, the *cardinality bounded portfolio selection*, was fully implemented within a branch-and-bound framework. Our presentation of Proximal-ACCPM focuses on the necessary information for an efficient implementation. It

also includes recent extensions, in particular an explicit treatment of second-order information when this information is available. The five selected examples have been reported in the literature. They are genuinely very large-scale problems. The first two are solved using a classical transformation known as Lagrangian relaxation. The transformed problem has much smaller dimension, thousands of variables instead of millions, but one can only collect information about it via a first-order oracle. It is shown that Proximal-ACCPM is powerful enough to solve huge instances of these problems. The third application fully exploits the concept of oracle based optimization to organize a dialog between two large-scale models that have totally different natures, a techno-economic model and a large-scale simulator of complex physics and chemistry processes. The exchanges between the two models are performed through few variables and each model is treated as a first-order oracle vis-à-vis these variables. These oracles, and especially the simulator, are computationally costly. The last example illustrates the use of Proximal-ACCPM within a branch-and-bound mechanism to solve a given MIP problem.

To make the OBO approach successful, one needs a method that keeps the number of calls to the oracles as low as possible. Proximal-ACCPM aims to achieve that purpose in a generic approach. However, number of important aspects still need to be seriously addressed. We list some of them.

- ◇ The kernel of the Proximal-ACCPM method involves solving a linear system, which the solution is expected to be accurate enough to yield a useful search direction for the next query point. As we get close to the boundaries or to the optimal solution, the principal matrix of the linear system becomes ill-conditioned, thus making difficult the computation of the required solution. This severely increases the associated computational cost, unless we chose to sacrifice the accuracy, which will extend the number of iterations to the solution. Thus, it is important to carefully address this issue, which belongs to the more general topic of solving ill-conditioned linear systems. However, there is probably a way to exploit the specific structure of the principal matrix in this case.

- ◇ The query point generator of the cutting planes method looks for a guess within a localization set defined by the set of cuts accumulated from the start. A good management of those cuts is crucial. Indeed, their number linearly increases with the number of iterations. If the dimension of the problem is huge, or if we have already performed a large number of iterations, the corresponding memory volume to keep the cuts will become significant, and this will slowdown the global memory efficiency. One way to fight against this problem is to eliminate redundant cuts, or to keep the minimal set of the cuts that correspond to the same (or equivalent) localization set. Doing this is not trivial as there are many valid configurations. Another way is to aggregate the cuts instead of eliminating them. We could also weight the cuts according to their impact on the localization set. All theses have to be studied deeply, at least from an experimental basis. However, we need to be careful with approach as we could destroy the coherence of the localization set, this either delay the convergence or completely diverge.

- ◇ Cutting planes methods are iterative, and the convergence is monitored by the calculation of the gap between the best solution found and the estimated lower bound (ideally the optimal objective value, but we don't have it). The process converges if: (a) the gap is below the tolerance parameter; (b) we have reached the maximum number of iterations; (c) a null gradient is provided by the oracle; (d) an incoherent information is

provided or calculated; (e) an unexpected critical hardware/system issue has occurred. The main focus here is the lower bound estimation. In Proximal-ACCPM, this is obtained from the localization set (the cuts + the objectives), thus could become heavy and inaccurate over the time (again, because of the large number of collected cuts and their relative layout). If the estimation of the lower bound is bad, we will perform a lot of additional iterations or never converge (even if we should, either because we are already at the optimum or there is no further improvement). It is therefore important to address this problem and provide a more robust routine.

◇ Concerning the newton system that is solved at iteration to get the search direction for the next query point, it could be very useful to find a way to work through less costly updates. Indeed, at each iteration, the matrix of generated cuts A is updated to $[A, u]$, where u is the new cut, then we solve a linear system using a matrix of the form

$$A \times diag(s^2) \times A^T, \tag{2.72}$$

where s is the vector of slack variables. It is quite frustrating to solve this system from scratch at each time. Indeed, the principal matrix (2.72) has a suitable form for a direct Cholesky factorization. One could imagine a way to keep on such a factorization, with the hope that this could be obtained by means of efficient updates (i.e. of a quadratic complexity instead of cubic as the whole factorization). The current state-of-the-art in matrix computation does not provides such a routine. Thus, this need to be investigated.

◇ For the branch-and-bound, we have provided a basic implementation illustrated on the *cardinality bounded portfolio selection* in order to test the effectiveness of Proximal-ACCPM for solving MIP problems. However, there are more sophisticated generic frameworks for the branch-and-bound associated with continuous optimization solvers [28]. An efficient connexion of Proximal-ACCPM with existing branch-and-bound solvers need to be studied.

◇ Proximal-ACCPM has only a sequential implementation. In order to take advantage of supercomputers, we need to investigate on its parallel implementation. This should be the occasion to consider the design of a parallel scalable branch-and-bound framework. Branch-and-bound is likely to yield an irregular computation scheme with an unpredictable path to the solution, thus making very challenging for efficient parallelization, especially on large-scale supercomputers. Among critical issues, we mention: *heavy synchronization, irregular communication pattern, huge amount of memory to handle the generated cuts*, and *load unbalanced*.

Acknowledgements. The work was partially supported by the Swiss NSF (Fonds National Suisse de la Recherche Scientifique, grant # 12-57093.99).

Bibliography

[1] C. J. Adcock and N. Meade, *A simple algorithm to incorporate transaction costs in quadratic optimization*, European Journal of Operational Research, **7**, 85-94, 1994.

[2] E. Agullo, B. Hadri, H. Ltaief and J. Dongarra, *Comparative study of one-sided factorizations with multiple software packages on multi-core hardware*,SC?09: International Conference for High Performance Computing, 2009

[3] E. Agullo, J. Dongarra, B. Hadri, J. Kurzak, J. Langou, J. Langou,

H. Ltaief, P. Luszczek, and A. YarKhan, *PLASMA: Parallel Linear Algebra Software for Multicore Architectures, Users? Guide*, `http://icl.cs.utk.edu/plasma/`, 2012.

[4] E. D. Andersen, J. Gondzio, Cs. Mészáros, and X. Xu, *Implementation of interior point methods for lardge scale linear programming*, T. Terlaky (Ed), Interior-point Methods of Mathematical Programming, Kluwer Academic Publishers, pp. 189-252, 1996.

[5] Applegate, D.L., Bixby, R.E., Chvatal, V., Cook, W.J., *The Traveling Salesman Problem - A Computational Study*, Princeton Series in Applied Mathematics, 2006.

[6] F. Babonneau and J.-P. Vial. ACCPM with a nonlinear constraint and an active set strategy to solve nonlinear multicommodity flow problems. Technical report, Logilab, Hec, University of Geneva, June 2005.

[7] F. Babonneau, C. Beltran, A. Haurie, C. Tadonki, and J.-P. Vial, *Proximal-ACCPM: a versatile oracle based optimization method*, in Optimisation, Econometric and Financial Analysis (E. J. Kontoghiorghes, ed.), vol. 9 of Advances in Computational Management Science, Springer Verlag, 2006.

[8] F. Babonneau, O. du Merle, and J.-P. Vial. Solving large scale linear multicommodity flow problems with an active set strategy and Proximal-ACCPM. *Operations Research*, 54(1):184–197, 2006.

[9] F. Babonneau, *Solving the multicommodity flow problem with the analytic center cutting plane method*, PhD thesis, University of Geneva, `http://archive-ouverte.unige.ch/unige:396`, 2006.

[10] Bader, D.A., Hart, W.E., Phillips, C.A., *Parallel Algorithm Design for Branch and Bound*, In: Greenberg, H.J. (ed.) Tutorials on Emerging Methodologies and Applications in Operations Research, ch. 5, Kluwer Academic Press, Dordrecht, 2004.

[11] S. Balay, J. Brown, K. Buschelman, V. Eijkhout, W. Gropp, D. Kaushik, M. Knepley, L. Curfman McInnes, B. Smith, and H. Zhang, *PETSc Users Manual. Revision 3.2*, Mathematics and Computer Science Division, Argonne National Laboratory, September 2011

[12] Barnhart et. al. , *Branch and Price : Column Generation for Solving Huge Integer Programs*,Operation Research, Vol 46(3), 1998.

[13] C. Barnhart, A. M. Cohn, E. L. Johnson, D. Klabjan, G. L. Nemhauser, P. H. Vance, *Airline Crew Scheduling*, Handbook of Transportation Science International Series in Operations Research & Management Science Volume 56, pp 517-560, 2003.

[14] C. Barnhart, E.L. Johnson, G.L. Nemhauser, M.W.P. Savelsbergh, and P.H. Vance. Branch-and-price: Column generation for solving huge integer programs. *Operations Research*, 46(3):316–329, 1998.

[15] C. Beltran, C. Tadonki, J.-Ph. Vial, *Semi-Lagrangian relaxation* , Computational Management Science Conference and Workshop on Computational Econometrics and Statistics, Link, Neuchatel, Switzerland, April 2004 .

[16] C. Beltran, C. Tadonki, and J.-P. Vial. Semi-lagrangian relaxation. Technical report, Logilab, HEC, University of Geneva, 2004.

[17] J. F. Benders, *Partitioning procedures for solving mixed-variables programming problems*, Numerische Mathematik 4(3), pp. 238?252, 1962.

[18] J. F. Benders. Partitioning procedures for solving mixed-variables programming problems. *Computational Management Science*, 2:3–19, 2005. Initially appeared in *Numerische Mathematik*, 4: 238-252, 1962.

[19] H. Y. Benson, D. F. Shanno, R.J. Vanderbei, *Interior-point methods for convex nonlinear programming: jamming and comparative numerical testing*, Op. Res. and Fin. Eng., ORFE-00-02- Princeton University, 2000 .

[20] C. Berger, R. Dubois, A. Haurie, E. Lessard, R. Loulou, and J.-P. Waaub. Canadian MARKAL: An advanced linear programming system for energy and environmental modelling. *INFOR*, 30(3):222–239, 1992.

[21] D. Bienstock, *Computational study of a family of mixed-integer quadratic programming problems*, Math. Prog. **74**, 121-140, 1996.

[22] L. S. Blackford, J. Choi, A. Cleary, E. D'Azevedo, J. Demmel, I. Dhillon, J. Dongarra, S. Hammarling, G. Henry, A. Petitet, K. Stanley, D. Walker, R. C. Whaley*ScaLAPACK*, `http://www.netlib.org/scalapack`, 2012.

[23] J. Blomwall, *A multistage stochastic programming algorithm suitable for parallel computing*, Parallel Computing, 29, 2003.

[24] R. A. Bosh and J.A. Smith, *Separating Hyperplanes and the Authorship of the Disputed Federalist Papers*, American Mathematical Monthly, Volume 105, No 7, pp. 601-608, 1995.

[25] Bosilca, G., Bouteiller, A., Danalis, A., Faverge, M., Haidar, H., Herault, T., Kurzak, J., Langou, J., Lemariner, P., Ltaief, H., Luszczek, P., YarKhan, A., Dongarra, J. *Distributed Dense Numerical Linear Algebra Algorithms on Massively Parallel Architectures: DPLASMA*, University of Tennessee Computer Science Technical Report, UT-CS-10-660, Sept. 15, 2010.

[26] S. Boyd , J. Mattingley, *Branch and Bound Methods*, Notes for EE364b, Stanford University, Winter 2006-07 (`http://see.stanford.edu/materials/lsocoee364b/17-bb_notes.pdf`).

[27] O. Briant and D. Naddef. The optimal diversity management problem. *Operations research*, 52(4), 2004.

[28] O. Briant, C. Lemarchal,K. Monneris,N. Perrot,C. Tadonki,F. Vanderbeck,J.-P. Vial,C. Beltran,P. Meurdesoif, *Comparison of various approaches for column generation*, Eigth Aussois Workshop on Combinatorial Optimization, 5-9 january 2004.

[29] A. Buttari, J. Langou, J. Kurzak, and J. Dongarra, *A class of parallel tiled linear algebra algorithms for multicore architectures*, Parallel Computing 35: 38-53, 2009.

[30] R. H. Byrd, et al., *Parallel global optimization for molecular configuration problem*, in Proceedings of the 6th SIAm Conference on Parallel Processing for Scientifi Computation, SIAM, Philadelphia, 1993.

[31] R. H. Byrd, et al., *Parallel global optimization: numerical methods, dynamic scheduling methods, and application to molecular configuration*, in B. Ford and A. Fincham (Eds), Parallel Computation, Oxford University Press, Oxford, pp. 187-207, 1993 .

[32] CPLEX, http://www.ilog.com/products/cplex/

[33] D. Carlson, A. Haurie, J.-P. Vial, and D.S. Zachary. Large scale convex optimization methods for air quality policy assessment. *Automatica*, 40:385–395, 2004.

[34] T. J. Chang, N. Meade, J. E. Beasley, Y. M. Sharaiha, *Heuristics for cardinality constrained portfolio optimization*, Computers & Operation Research, 27, pp. 1271-1302, 2000.

[35] V. Chvátal, *Hard Knapsack Problems*, Operations Research, Vol. 28(6),pp. 1402-1411,1980.

[36] V. Chvatal, *Linear Programming*, W. H. Freeman Compagny, Series of Books in the Mathematical Sciences, 1983.

[37] A. Clementi, J. D. P. Rolim, and E. Urland, *Randomized Parallel Algorithms*, LLNCS, A. Ferreira and P. Pardalos (Eds), pp. 25-48, 1995.

[38] Constantinides G. M. and Malliaris A.G., *Portfolio theory*, Finance ed R.A. Jarrow, V. Maksimovic and W. T. Ziemba, Elsevier-Amsterdam, 1-30, 1995.

[39] W. J. Cook, W. H. Cunningham, W. R. Pulleyblank, A. Schrijver, *Combinatorial Optimization*, John Wiley & Sons, 1998.

[40] T. Crainic, B. Le Cun, and C. Roucairol, *Parallel Branch-and-Bound Algorithms*, In: Talbi, E. (ed.) Parallel Combinatorial Optimization, ch. 1, Wiley, Chichester 2006.

[41] M. D'Apuzzo and M. Marino, *Parallel computation issued of an interior point method for solving large bound-constrained quadratic programming problems*, Parallel Computing, 29, 2003.

[42] M. D'Apuzzo, et al., *A parallel implementation of a potential reduction algorithm for box-constrained quadratic programming*, in LLNCS pp. 839-848, Europar2000, Spriger-Verlag, Berlin, 2000.

[43] M. D'Apuzzo, et al., *Nonlinear optimization: a parallel linear algebra standpoint*, Handbook of Parallel Computing and Statistics, E. J. Kontoghiorges (Ed.), New-York, 2003.

[44] M. D'Apuzzo, et al.,*Parallel computing in bound constrained quadratic programming*, Ann. Univ. Ferrara-Sez VII-Sc. Mat. Supplemento al XLV pp. 479-491, 2000.

[45] A. De Bruin, G. A. P. Kindervater, H. W. J. M. Trienekens , *Towards and abstract parallel branch and bound mahcine*, LLNCS, A. Ferreira and P. Pardalos (Eds), pp. 145-170, 1995.

[46] Z. Degrave and M. Peeters, *Benchmark Results for the Cutting Stock and Bin Packing Problem*, Research Report No 9820 of the Quantitative Methods Group, Louvain, Belgique, 1998.

[47] L. Drouet, C. Beltran, N.R. Edwards, A. Haurie, J.-P. Vial, and D.S. Zachary. An oracle method to couple climate and economic dynamics. In A. Haurie and L. Viguier, editors, *Coupling climate and economic dynamics*. Kluwer (to appear), 2005.

[48] L. Drouet, N.R. Edwards, and A. Haurie. Coupling climate and economic models in a cost-benefit framework: A convex optimization approach. *Environmental Modeling and Assessment*, to appear in 2005.

[49] I. S. Duff, H. A. VanDer Vorst, *Developments and trends in the parallel solution of linear systems*, Parallel Computing 25, pp. 13-14, 1999.

[50] C. Durazzi and V. Ruggiero, *Numerical solution of special linear and quadratic programs via a parallel interior-point method*, Parallel Computing, 29, 2003.

[51] O. Du Merle and J.-P. Vial. Proximal ACCPM, a cutting plane method for column generation and Lagrangian relaxation: application to the p-median problem. Technical report, Logilab, University of Geneva, 40 Bd du Pont d'Arve, CH-1211 Geneva, Switzerland, 2002.

[52] Fengguang, S., Tomov, S., Dongarra, J., *Efficient Support for Matrix Computations on Heterogeneous Multi-core and Multi-GPU Architectures*, University of Tennessee Computer Science Technical Report, UT-CS-11-668, June 16, 2011.

[53] A. Ferreira and P. M. Pardalos (Eds.), *Solving Combinatorial Optimization Problems in Parallel*, LLNCS-Springer 1054, 1995 .

[54] A. Ferreira and P. M. Pardalos, *Parallel Processing of Discrete Optimization Problems*, DIMACS Series Vol. 22, American Mathematical Society, 1995.

[55] M. C. Ferris and T.S. Munson, *Interior Point Methods for Massive Support Vector Machines*, Cours/Sminaire du 3^e cycle romand de recherche oprationnelle, Zinal, Switzerland, march 2001.

[56] M. C. Ferris, J. D. Horn, *Partitioning mathematical programs for parallel solution*, Mathematical Programming 80, PP. 35-61, 1998.

[57] Ferris, M., *GAMS: Condor and the grid: Solving hard optimization problems in parallel*, Industrial and Systems Engineering, Lehigh University, 2006.

[58] L. G. Fishbone and H. Abilock. MARKAL, a linear programming model for energy systems analysis: Technical description of the BNL version. *International Journal of Energy Research*, 5:353–375, 1981.

[59] E. Fragnière and A. Haurie. A stochastic programming model for energy/environment choices under uncertainty. *International Journal of Environment and Pollution*, 6(4-6):587–603, 1996.

[60] E. Fragnière and A. Haurie. MARKAL-Geneva: A model to assess energy-environment choices for a Swiss Canton. In C. Carraro and A. Haurie, editors, *Operations Research and Environmental Management*, volume 5 of *The FEEM/KLUWER International Series on Economics, Energy and Environment*. Kluwer Academic Publishers, 1996.

[61] M. R. Garey and D. S. Johnson, *Computers and Intractability: A Guide to the Theory of NP-Completeness*, Morgan Kaufmann, 1979.

[62] C. Gatu and E. J. Kontoghiorghes, *Parallel aalgorithms for computing all possible subset regression models using the QR decomposition*, Parallel Computing, 29, 2003.

[63] M. Gengler, *An introduction to parallel dynamic programming*, LLNCS, A. Ferreira and P. Pardalos (Eds), pp. 86-114, 1995.

[64] A. M. Geoffrion. Lagrangean relaxation for integer programming. *Mathematical Programming Study*, 2:82–114, 1974.

[65] J.-L. Goffin and J.-P. Vial. Convex nondifferentiable optimization: A survey focussed on the analytic center cutting plane method. *Optimization Methods and Software*, 174:805–867, 2002.

[66] J.-L. Goffin and J.-P. Vial. Shallow, deep and very deep cuts in the analytic center cutting plane method. *Mathematical Programming*, 84:89–103, 1999.

[67] J. L. Goffin, A. Haurie, and J. P. Vial, *Decomposition and nondifferentiable optimization with the projective algorithm* Management Science, **37**, 284-302.

[68] J.-L. Goffin, Z. Q. Luo, and Y. Ye. Complexity analysis of an interior point cutting plane method for convex feasibility problems. *SIAM Journal on Optimization*, 69:638–652, 1996.

[69] J. Gondzio, A. Grothey, *Parallel Interior Point Solver for Structured Quadratic Programs: Application to Financial Planning Problems*, RR MS-03-001, School of Mathematics, University of Edinburgh, 2003.

[70] J. Gondzio, O. du Merle, R. Sarkissian and J.P. Vial, *ACCPM - A Library for Convex Optimization Based on an Analytic Center Cutting Plane Method*, European Journal of Operational Research, 94, 206-211, 1996.

[71] A. Grama and V. Kumar, *State-of-the-Art in Parallel Search Techniques for Discretes Optimization Problems*, Personnal communication, 1993 .

[72] M. Guignard and S. Kim. Lagrangean decomposition: a model yielding stronger Lagrangean bounds. *Mathematical Programming*, 39:215–228, 1987.

[73] A. Gupta, G. Karypis, V. Kumar, *A highly scalable parallel algorithm for sparse matrix factorization*, IEEE TPDS 8(5), pp. 502-520.

[74] N. H. Hakansson, *Multi-period mean-variance analysis: Toward a theory of portfolio choice*, Journal of Finance, **26**, 857-884, 1971.

[75] J. Han and M. Kamber, *Data Mining: Concept and Techniques*, Morgan Kaufmann Publishers, 2000.

[76] P. Hansen, N. Mladenovic, and D. Perez-Brito. Variable neighborhood decomposition search. *Journal of Heuristics*, 7:335–350, 2001.

[77] A. Haurie, J. Kübler, A. Clappier, and H. Van den Bergh. A metamodeling approach for integrated assessment of air quality policies. *Environmental Modeling and Assessment*, 9:1–122, 2004.

[78] J. L. Houle, W. Cadigan, S. Henry, A. Pinnamanenib, S. Lundahlc, *Database Mining in the Human Genome Initiative*, `http://www.biodatabases.com/whitepaper01.html`

[79] B. Hunsaker, C. A. Tovey, *Simple lifted cover inequalities and hard knapsack problems*, Discrete Optimization 2(3), pp. 219-228, 2005.

[80] N. J. Jobst, M. D. Horniman, C. A. Lucas, and G. Mitra, *Computational aspects of alternative portfolio selection models in the presence of discrete asset choice constraints*, Quantitative finance, Vol. 1, p. 1-13, 2001.

[81] L. Jooyounga, et al., *Efficient parallel algorithms in global optimization of potential energy functions for peptides, proteins, and crystals*, Computer Physics Communications 128 pp. 3999-411, 2000.

[82] Judea Pearl, *Heuristics-Intelligent Search Strategies for Computer Problem Solving*, Addison-Wesley, Reading, MA, 1984.

[83] O. Kariv and L. Hakimi. An algorithmic approach to network location problems. ii: the p-medians. *SIAM Journal of Applied Mathematics*, 37(3):539–560, 1979.

[84] J. Kepner, *MatlabMPI*, `http://www.ll.mit.edu/MatlabMPI/`

[85] M. Kleinberg, C.H. Papadimitriou, and P. Raghavan, *Segmentation Problems*, ACM Symposium on Theory of Computing, 1998, pp. 473-482.

[86] E. K. Lee and J. E. Mitchell , *Computational experience of an interior-point SQP algorithm in a parallel branch-and-bound franmework*, Proc. High Perf. Opt. Tech., 1997.

[87] C. Lemaréchal, *Lagrangian relaxation*, M. Junger and D. Naddef (Eds.): Computat. Comb. Optimization, LNCS 2241, pp. 112?156, 2001.
`http://link.springer.com/content/pdf/10.1007%2F3-540-45586-8_4`

[88] C. Lemaréchal. Nondifferentiable optimization. In G.L. Nemhauser, A.H.G Rinnooy Kan, and M.J. Todd, editors, *Handbooks in Operations Research and Management Science*, volume 1, pages 529–572. North-Holland, 1989.

[89] R. M. Lewis and V. J. Torczon, *Pattern search methods for linearly constrained minimization*, SIAM Journal of Optimization, 10, pp. 971-941, 2000.

[90] D. Li and W. L. Ng, *Optimal dynamic portfolio selection: Multi-period mean-variance formulation*, Math. Finance **10**, 387-406, 2000.

[91] MOSEK, http://www.mosek.com/.

[92] O. L. Mangasarian, R. Setino, and W. Wolberg, *Pattern Recognition via linear programming: Theory and Applications to Medical Diagnosis*, 1990.

[93] O. L. Mangasarian, W.N. Street, and W.H. Wolberg, *Breast Cancer Diagnosis and prognosis via linear programming*, Operation research, Vol. 43, No. 4, July-August 1995, pp. 570-577.

[94] O. L. Mangasarian, *Linear and Non-linear Separation of Patterns by linear programming*, Operations Research, 13, pp. 444-452.

[95] R. Mansini and M. G. Speranza, *Heuristic algorithms for the portfolio selection poblem with minimum transaction lots*, Eur. Jour. Op. Res., **114**, 219-223, 1999.

[96] H. Markowitz, *Portfolio Selection: Efficient Diversification of Investment*, John Wiley & Sons, New-York, 1959.

[97] H. Markowitz, *Portfolio Selection*, The Journal of Finance **1**, 77-91, 1952.

[98] A. Migdalas, G. Toraldo, and V. Kumar, *Nonlinear optimization and parallel computing*, Parallel Computing 29, pp. 375-391, 2003.

[99] J. Mossin, *Optimal multiperiod portfolio policies*, J. Business, **41**, 215-229, 1968.

[100] Y. Nesterov and A. Nemirovsky. *Interior Point Polynomial Algorithms in Convex Programming: Theory and Applications*. SIAM, Philadelphia, Penn., 1994.

[101] Y. Nesterov and J.-P. Vial. Homogeneous analytic center cutting plane methods for convex problems and variational inequalities. *SIAM Journal on Optimization*, 9:707–728, 1999.

[102] Y. Nesterov. Complexity estimates of some cutting plane methods based on the analytic center. *Mathematical Programming*, 69:149–176, 1995.

[103] Y. Nesterov. *Introductory Lectures on Convex Optimization, a Basic Course*, volume 87 of *Applied Optimization*. Kluwer Academic Publishers, 2004.

[104] P.S. Pacheco, *Parallel Programming with MPI*, Morgan Kaufmann, 1997.

[105] C. H. Papadimitriou and K. Steiglitz, *Combinatorial optimization: algorithms and complexity*, Prentice-Hall 1982.

[106] P. M. Pardalos, A. T. Phillips, and J. B. Rosen, *Topics in Parallel Computing in Mathematical Programming*, Science Press, 1993.

[107] P. M. Pardalos, M. G. C. Resende, and K. G. Ramakrishinan (eds), *Parallel Processing of Discrete Optimization Problems*, DIMACS Series Vol. 22, American Mathematical Society, 1995.

[108] Paul A. Jensen and Jonathan F. Bard, *Operations Research - Models and Methods*, John Wiley and Sons , 2003.

[109] Per. S. Lauren, *Parallel heuristic search - Introduction and new approach*, LLNCS, A. Ferreira and P. Pardalos (Eds), pp. 248-274, 1995.

[110] G. Reinelt. Tsplib, 2001. http://www.iwr.uni-heidelberg.de / groups / comopt / software / TSPLIB95.

[111] P. A. Samuelson, *Lifetime portfolio selection by dynamic stochastic programming*, Rev. Econ. Stat. **51**, 239-246, 1969.

[112] R. B. Schnabel, *A view of the limitation, opportunities, and challenges in parallel nonlinear optimization*, Parallel Computing 21(6), pp. 875-905, 1995.

[113] Scott Hamilton and Lee Garber, *Deep Blue's hardware-software synergy*, IEEE Computer, 30(10), pp. 29-35, 1997.

[114] Y. Shinano, T. Fujie, *ParaLEX: A Parallel Extension for the CPLEX Mixed Integer Optimizer*, Recent Advances in Parallel Virtual Machine and Message Passing Interface Lecture Notes in Computer Science Volume 4757, pp 97-106, 2007.

[115] S. L. Smith, R. B. Schnabel, *Centralized and distributed dynamic scheduling for adaptative parallel algorithms*, in P. Mehrotra, J. Saltz, R. Voight (Eds), Unstructured Scientific Computation on Scalable Multiprocessors, MIT Press, pp. 301-321, 1992.

[116] C. Tadonki and J.-P. Vial, *Efficient algorithm for linear pattern separation*, (to appear in) International Conference on Computational Science, ICCS04 (LNCS/Springer), Krakow, Poland, June 2004 .

[117] C. Tadonki, C. Beltran and J.-P. Vial , *Portfolio management with integrality constraints*, Computational Management Science Conference and Workshop on Computational Econometrics and Statistics, Link, Neuchatel, Switzerland, April 2004 .

[118] C. Tadonki, J.-P. Vial, *Efficient Algorithm for Linear Pattern Separation*, International Conference on Computational Science, ICCS04 (LNCS/Springer), Krakow, Poland, June 2004.

[119] C. Tadonki, *A Recursive Method for Graph Scheduling*, International Symposium on Parallel and Distributed Computing (SPDC), Iasi, Romania, July 2002

[120] C. Tadonki, *Parallel Cholesky Factorization*, Workshop on Parallel Matrix Algorithm and Applications (PMAA), Neuchatel, Switzerland, August 2000.

[121] E.-G. Talbi (Editor), *Parallel Combinatorial Optimization*,Wiley Series on Parallel and Distributed Computing,2006.

[122] S. Tomov R. Nath P. Du J. Dongarra, *MAGMA: Matrix Algebra on GPU and Multicore Architectures*, `http://icl.cs.utk.edu/magma`, 2012.

[123] R. A. Van de Geijn, *Using PLAPACK*, The MIT Press, 1997.

[124] Vance et. al. , *Using Branch-and-Price-and-Cut to solve Origin-Destination Integer Multicommodity Flow problems*,Operation Research, Vol 48(2), 2000.

[125] P. H. Vance, A. Atamturk, C. Barnhart, E. Gelman, and E. L. Johnson, A. Krishna, D. Mahidhara, G. L. Nemhauser, and R. Rebello, *A Heuristic Branch-and-Price Approach for the Airline Crew Pairing Problem*, 1997.

[126] M.S. Viveros, J.P. Nearhos, M.J. Rothman, *Applying Data Mining Techniques to a Health Insurance Information System*, 22nd VLDB Conference, Mumbai(Bombay), India,1996, pp. 286-294.

[127] B. W. Wah, G.-J. Li, and C. F. Yu, *Multiprocessing of combinatorial search problems*, IEEE Computer, June 1985.

[128] R. C. Whaley, et al., *Automated empirical optimizations of software and the ATLAS project*, Parallel Computing 27, pp. 3-35, 2001.

[129] M. R. Young, *A minimax portfolio selection rule with linear programming solution*, Management Science **44**, 673-683, 1992.

[130] G. Zanghirati and L. Zanni, *A parallel solver for large quadratic programs in training support vector machines*, Parallel Computing, 29, 2003.

[131] T. Zariphoulou, *Investment-consumption models with transactions costs and Marko chain parameters*, SIAM J. Control Optim **30**, 613-636, 1992.

[132] X. Y. Zhou and D. Li, *Continuous-Time mean-variance portfolio selection: A stochastic LQ framework*, Appl. Math. Optim. **42**, 19-33, 2000.

[133] *CPLEX*, `http://www.ilog.com/products/cplex`

[134] *LOGO*, `http://www.orfe.princeton.edu/ loqo/`

[135] *MOSEK*, `http://www.mosek.com/`

Chapter 3

Accelerated computing

3.1 Abstract

The current chapter presents our contributions in the field of *accelerated computing*. A description of our achievements is provided [23, 25, 24, 20], surrounded by technical discussions about critical points and the perspectives. Accelerated computing is an important notion one should keep in mind when it comes to future generations of supercomputers. The basic idea is to offload highly regular tasks to one or several accelerators associated to the main CPU. This requires the programmer to organize the computation as a combination of regulars kernels (to be executed on the accelerators) and explicitly specify the management of data transfers to/from the host. Accelerators are indeed very fast, but the cost of loading and storing data is a performance bottleneck in most cases. The common way to overcome this issue is to overlap computation and data exchanges whenever possible, and follow the strict technical recommendations for efficient memory accesses. We illustrate this topic through three different case studies on the CELL Broadband Engine. The most important aspect here is not the device itself, but our methodology instead.

3.2 The CELL Broadband Engine

The CELL Architecture, also known as the Cell Broadband Engine, grew from the need to provide power-efficient and cost-effective high-performance processing for a wide range of applications. Cell is a multi-core chip that consists of an IBM 64-bit Power Architecture core, augmented with eight specialized co-processors based on a specific single-instruction multiple-data (SIMD) architecture called Synergistic Processor Unit (SPU), which is for data-intensive processing. The system is integrated by a coherent on-chip bus and can deliver up to 256 GFlops in single-precision and 26/108 GFlops in double-precision (with 8 SPUs) [17].

Figure 3.1: Cell Block Diagram

The CELL really provides a significant processing power with a low power consumption, thus making it a good candidate as a computing node for a modern supercomputer. Indeed, the 2008 world fastest supercomputer, the *IBM Roadrunner*, is made with PowerXCell 8i (and conventional AMD Opteron processors). Moreover, the Roadrunner was the first supercomputer to deliver a sustained *petaflop* performance. Note that the current world fastest supercomputer (Titan-CRAY) is also made up with accelerated nodes (using GPUs). At least for these reasons, accelerated computing deserves a close attention.

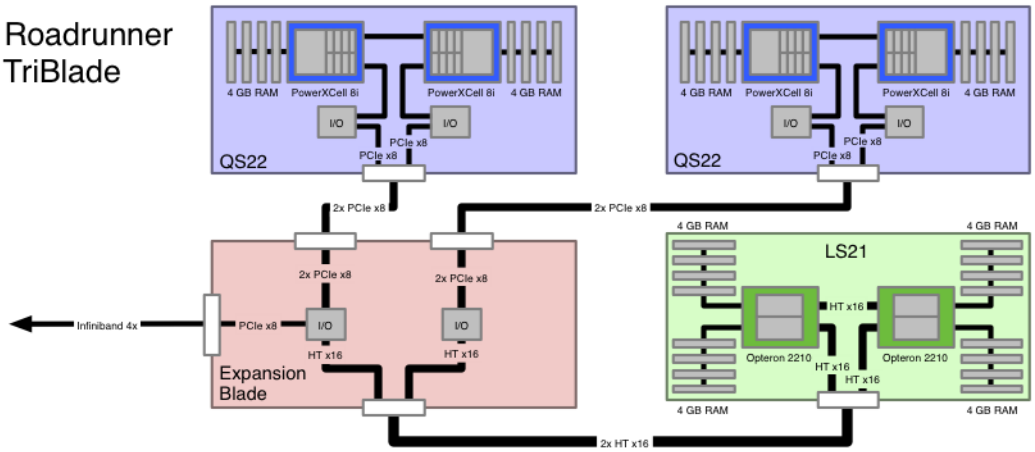


Figure 3.2: Roadrunner node

Another good illustration is the QPACE architecture, mainly devoted to Lattice Quantum ChromoDynamics (LQCD). In [18], QPACE architecture is presented as a new, scalable Lattice QCD machine based on the IBM PowerXCell 8i, with the following design highlights:

- ◇ Fast commodity processor = IBM PowerXCell 8i
- ◇ FPGA directly attached to processor
- ◇ LQCD optimized torus network (custom network)
- ◇ Custum system design
- ◇ Novel, cost-efficient liquid cooling system
- ◇ Very power efficient architecture
- ◇ Two installations with an aggregate performance of 200/400 TFlops (DP/SP)
- ◇ Good sustained performance of O(20-30%) for key LQCD kernels → O(10-15) TFlops/rack (SP)

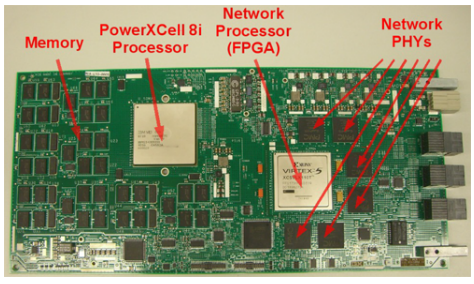


Figure 3.3: QPACE node-card

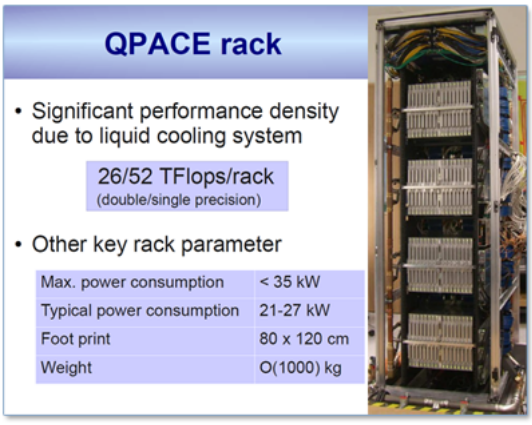


Figure 3.4: QPACE rack data

The need of a custom network for the QPACE architecture illustrates the fact that having efficient computing nodes exacerbates the need of a faster interconnect.

The CELL processor has proved efficient on a wide range of stream processing applications. Under the motivation of reported results and from its potential capabilities, its effective/prospective use has been extended to traditional high performance computing topics. The price to achieve an efficient program on the CELL is the underlying programming effort due to specific hardware and software constraints. Indeed, although the strong potential of the CELL, as other accelerators like the GPU, harvesting a significant fraction of the theoretical peak performance is a difficult programming task. In fact, accelerators are made for highly regular computation, preferably operating on single precision numbers. Thus, on a standard application, there are number of performance issues that need to be addressed very carefully. Among the important aspects, we cite:

- ◇ *data alignment:* program running on the CELL need to have aligned data and align memory references. This is required both for the computation on the SPEs and for PPE↔SPEs memory accesses.
- ◇ *local store size:* each SPE has a rather small local memory (called *local store*). Thus, lot of block memory accesses are required when operating on large arrays. In addition, the program that has to be executed on the SPE should be sufficiently lightweight to fit into the local store while leaving enough space for the minimum chunk of input/output data.
- ◇ *double precision penalty:* whatever the version of the hardware, operating with double precision numbers severely affect the sustained performance of the CELL. The programmer has to deal with a compromise between the real need of a double precision computation and the associated performance slowdown. This aspect was improved in the latest version of the CELL (PowerXCell 8i), but the penalty remains.
- ◇ *different level of parallelism:* a coarse grain parallelism is applied when distributing the tasks among the SPEs, and with a SPE, a fine grained parallelism is considered through SIMD instructions.
- ◇ *memory hierarchy:* this is one of the most important point when programming the CELL. The memory organization is quite particular and data need to be loaded/sorted from/to the main memory located into the main CPU, which orchestrates the overall computation process. Access to the main memory, called *Direct Memory Access* (DMA), has very strict rules. Some of them are mandatory, while other are related to performance purposes. The main technical consequence is that the programmer has to manually adapt the data layout accordingly, which is likely to be a tedious task. We address this issue in the section 3.3 and present our related contribution.

3.3 Generic DMA Routine

3.3.1 Introduction

On parallel and/or accelerated computing systems, because the communication latency is likely to dominate, the cost of communicating a single data element is only marginally different from that for a "packet" of data. Therefore, it is necessary to combine the results of a number of elementary computations, and send these results together. This is typically achieved by a technique called *supernode partitioning* [9] (also called *iteration space tiling*,

loop blocking, or *clustering*), where the computation domain is partitioned into (typically parallelepiped shaped) identical "blocks" and the blocks are treated as atomic computations. Such a clustering, which can be applied at various algorithmic level, has also proven to be a good way for a systematic transition from a fine-grained parallelism to a coarse-grained parallelism [2].

Although tiling [34] is a well known strategy, even used on uniprocessors to exploit hierarchical memories [8], most compilers are only able of tiling rather simple programs (perfectly nested loops with *uniform* dependences). Thus, producing a tiled version of a given program is likely to be a manual task.

The case of the CELL is rather special. Indeed, transferring data between the PPU and the SPEs (DMA) is subject to specific constraints which dictate the shape of any tiling approach. From the performance point of view, this might impede the use the optimal tile shape for instance. From the design point of view, accessing and internal block (whose any of its dimensions does no equal that of its container) would require a tedious programming effort. Such a generic tiling is essential for block pivoting algorithms for instance. This work addresses the problem and provides a framework that makes it easier to implement a tiled model on the CELL. Indeed, using our routines, the programmer just need to provide the references of the tile (pointer and dimensions) whatever their shape and location. We perform the required DMAs with the necessary pre-processing and/or post-processing (if any). In addition to be able to perform the task seamlessly, we achieve the requested DMA with a negligible software latency in any cases.

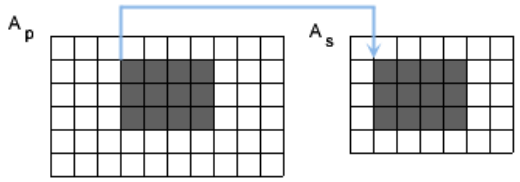
3.3.2 DMA rules and description of the need

A DMA of V bytes from location a to location b must fulfill one the following requirements:

(1) $(V \in \{1, 2, 4, 8\}) \quad \wedge \quad (a \bmod 16 = b \bmod 16)$
(2) $(V = 16q,\ q \leq 1024) \quad \wedge \quad (a \bmod 16 = b \bmod 16 = 0)$

A *list DMA* follows the same rules, applied to each item within the list, and can handle up to 2048 transfers in a single request (treated sequentially). From the PPE to a given SPE, a *list DMA* gathers all the data from each individual DMA within the list to the target location. From a given SPE to the PPE, a *list DMA* scatters the source array into the different locations indicated by each individual DMA. For the programmer, memory addresses are processed as variables of type `unsigned int`.

The problem we want to solve can be stated as follows. Given A_p, a $n_p \times m_p$ matrix located into the main memory (PPE), and A_s, a $n_s \times m_s$ matrix located into the local store (SPE), how can we copy the $a \times b$ submatrix of A_p located at (i_p, j_p) into A_s at location (i_s, j_s). Figure 3.5 depicts the task.



Main memory: $n_p = 6,\ m_p = 10,\ i_p = 2,\ j_p = 4$
Local store: $n_s = 5,\ m_s = 7,\ \ i_s = 2,\ j_s = 2$
$a = 3,\ \ b = 4$

Figure 3.5: Generic DMA pattern

Performing the DMA as expressed in figure 3.5 raises the following issues:

1. The region to be transfered is not contiguous on memory, thus an explicit list DMA will be used most of the time to achieve the task.
2. One "row" is referenced by an misaligned memory address, thus the whole list DMA will not be executed. It is thus critical to have each row aligned, which is unlikely to happen by plain coincidence; an appropriate preprocessing is then needed.
3. One "row" has an (address, volume) pair which does not match the basic DMA rules (see (1)&(2) of section 3.3.2), thus the whole list DMA cannot be executed.
4. Either the source address or the target address is misaligned (this can occur even if the initial allocations were aligned). The DMA will be canceled.
5. The target region on the local store is out of the container limits. This commonly happens which boundary tiles. This will lead to data corruption.

Our goal is to overcome the above problems at the minimum processing time-cost, since the consequent (pre/post)processing is an overhead for the programmer. We now provide a complete formulation of each problematic configuration and describe our solution.

3.3.3 Description of our solution

The generic data structure we use to store the basic information concerning the main matrix A_p (located on the main memory) is the following:

```
struct spe_arg_t {
   unsigned int matrix;
   unsigned int nblines;
   unsigned int nbcols;
   unsigned int datatype_size;
   unsigned int my_ppe_buffer;
}__attribute__((aligned(16)));
```

One `spe_arg_t` variable is created par SPE, where

- ◇ `matrix` is a pointer to the main matrix (A_p on the PPE)
- ◇ `nblines` (resp. `nbcols`) is the height (resp. width) of the main matrix
- ◇ `datatype_size` is the size (in bytes) of each element of the matrix. We use this information to calculate the volume (in bytes) of the submatrix to be transfered and to perform necessary adjustments as we shall see.
- ◇ `my_ppe_buffer` is a pointer to the main memory region where the SPE should send back its results (if any).

Typically, the PPE initializes and sends one `spe_arg_t` variable to each of the SPEs. Next, the each SPE iterates on different blocks using our DMA routine and synchronizes with the PPE through mailboxes.

We now describe the work done to get (resp. send) a tile on the SPE (resp. to the PPE).

3.3.4 From the PPE to the SPE

Figure 3.6 synthesizes the routine executed by the SPE to load a tile from the main memory.

Figure 3.6: Tile transfer workflow

We now address each of the issues raised in section 3.3.2.

1. We check if the *width* of the submatrix equals that of its container (i.e. $b = m_p$). In that case, we just need a single DMA, since the submatrix is a contiguous block memory. Otherwise, we prepare a list DMA, where each DMA item is devoted to a row of the tile. Note that for a single DMA request, if the volume is greater than 16 KB, then we convert the request into an equivalent list DMA.

2. We inspect each DMA item to check for alignment and volume constraints. If the address p is not aligned, then we look for the greatest aligned address p' lower than p ($p' < p$). Typically, $p' = p - r$, where $r = p \bmod 16$. One can search for r iteratively among $1, 2, \cdots, 15$. Now, the volume to be transfered is $V + (p - p')$. Again we need to adjust that volume to the nearest aligned one (factor of 16), we denote V'.

3. Once a valid list DMA is prepared, we need to focus on where to store the data. If the previous checks did reveal some issues, then we first get the DMA data into a buffer, because a postprocessing is necessary to extract relevant data. This is also required if the target region in the local store is not contiguous (i.e. $b < m_s$). Indeed, a list DMA stores data in a contiguous way into the local store. We reshape the data while copying them from the buffer to the right destination.

In any of the above cases, our implementation is optimized in order to minimize the overhead. Technical details (conceptual improvements and programming strategies) are omitted for simplicity. The whole chain is provided through a seamless interface described by the following generic command

`spe_get_tile`$(A_p, n_p, m_p, i_p, j_p, a, b, A_s, n_s, m_s, i_s, j_s)$

3.3.5 From the SPE to the PPE

Since all of our DMAs are issued from the SPE, we need a synchronization mechanism to complete the postprocessing on the PPU (in a symmetric way as the PPE→SPE transfer). For this purpose, we chose to use the mailboxes. Once the DMA put from the SPE local store is completed (into the PPE buffer referenced by `ppe_buffer`), the SPE sends a mail to the PPE to indicate that its data are available. The PPE then copies the data from the corresponding buffer to the right destination (known at the PPE side only). We provide a PPE routine to perform such a copy/reshape from the buffer. The command to be issued from the SPE is the following:

`spe_put_tile`(`ppe_buffer`, $A_s, n_s, m_s, i_s, j_s, a, b$)

At completion, the buffer is automatically freed for future uses. Once this is done, the SPE

sends a mail to the PPE, who then executes the command
`ppe_put_tile`(`ppe_buffer`, $A_r, n_r, m_r, i_r, j_r, a, b$)
to store the block data back to the right location. We now provide some performance measurements of our procedure.

3.3.6 Performance measurements

We consider a 200×125 matrix of `unsigned int` (i.e. `datatype_size` $= 4$) on the main memory and a 100×125 matrix on the SPE. For $k = 1, 2, \cdots, 20$, we copy a tile of size 20×125 from $(k, 0)$ to $(k, 0)$. The container matrices are aligned in both sides. Thus, no additional processing is needed when k is a factor of 4 (because $125 * k * 4$ is a multiple of 16). Figure 3.7 displays the measured performances on a QS22 blade.

k	time(μs)	(pre/post)processing
0	15.020	0
1	22.888	1
2	16.928	1
3	17.166	1
4	12.875	0
5	17.881	1
6	17.166	1
7	18.120	1
8	12.875	0
9	16.928	1
10	16.955	1
11	18.120	1
12	12.875	0
13	16.928	1
14	16.928	1
15	17.166	1
16	12.875	0
17	16.928	1
18	16.928	1
19	18.120	1

Figure 3.7: Tiled DMA timings

$tile_h$	$tile_w$	time($10^{-3}s$)
4	1024	8.62
8	512	9.16
16	256	9.32
32	128	9.43
64	64	10.67
128	32	12.92
256	16	16.84
512	8	24.95

Figure 3.8: Tiled DMA timings

We see that our extra processing has a cost of 25% in average, which seems acceptable. In a context where computation significantly dominates, the overhead of our procedure will become quite negligible. The second experiment is performed with a 512×1024 matrix of `float`. We perform a tiled round trip with the entire matrix between the PPE and the SPE. Figure 3.8 provides the timings with various tile sizes (fixed volume, i.e. $tile_h \times tile_w = cste$). We see that, for a fixed tile volume, the performance variance with various possible tile shapes is relatively marginal.

3.4 The Harris corner detection algorithm

3.4.1 abstract

Real-time implementation of *corner detection* is crucial as it is a key ingredient for other image processing kernels like *pattern recognition* and *motion detection*. Indeed, *motion detection* requires the analysis of a continuous flow of images, thus a real-time motion detection would require the use of highly optimized subroutines. We consider a tiled implementation of the Harris corner detection algorithm on the CELL processor. The algorithm is a chain of convolution kernels and point-to-point matrix operations. The corresponding memory access pattern is a stencil, which is known to exacerbate cache misses. In order to reduce the consequent time overhead, tiling is a commonly considered way. When it comes to *image processing filters*, incoming tiles are overdimensioned to include their neighborhood, necessary to update boundary pixels. As the volume of this "extra data" depends on the tile shape (its dimensions), we need to seek a good tiling strategy. On the CELL, such an investigation is not directly possible with native DMA routines. We overcome the problem by using our previously described framework, which enhances the DMA mechanism to operate with non conventional requests. Based on this extension, we experiment various tile sizes and shapes on the CELL, thus trying to confirm our intuition on the optimal clustering.

3.4.2 Introduction

The common characteristic of image processing algorithms is the heavy use of convolution kernels. Indeed, the typical scheme is an iterative application of a stencil calculation at the pixel level. The fact that each output pixel is obtained from the corresponding input pixel and its periphery breaks any hope of regular memory accesses, thus making it hard to achieve a real-time performance implementation.

The *Harris* algorithm [7] for corner detection is an interesting case study application because it allows various implementations and different optimization strategies [20]. Among these possibilities, tiling [34] is potentially attractive as it can be naturally applied on top of any valid scheduling to improve memory performance. However, tiling on the CELL cannot be directly implemented because of data alignment constraints when using native DMA routines. Because of this constraint, tiles corresponding to contiguous memory region (full row tiles for instance) are used most of the time, thus preventing other choices for the tile shape.

Tile shape restriction is particularly frustrating with image processing operators because either it does not allow the use of a predicted optimal tile shape, or it acts as a runtime bottleneck. The later could occurs, for instance, with an image so large that the SPE local store cannot hold three of its entire rows (one active row plus its top and bottom neighborhoods). *Data alignment* is another critical requirement. In this work, we rely on a our framework, which provides a seamless way to deal with any tile shape. We study the effect of tiling and report experimental results driven by theoretical predictions. Our approach is more general an can be considered for any accelerated-based computation, the current illustration on the CELL BE attempts to validate our strategy. We now described three case studies from our contributions.

3.4.3 The Harris-Stephen algorithm

Harris and Stephen [7] interest point detection algorithm is an improved variant of the *Moravec*

corner detector [16], used in computer vision for feature extraction like *motion detection, image matching, tracking, 3D reconstruction* and *object recognition.* Figure 3.9 illustrates the use of the algorithm in the basic case of corner detection.

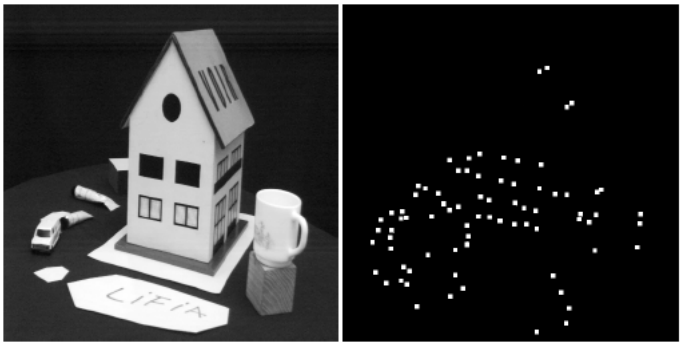


Figure 3.9: Illustration of the Harris-Stephens procedure

The algorithm is mainly a successive application of convolution kernels that globally implement a discrete form of an autocorrelation S, given by

$$S(x,y) = \sum_{u,v} w(u,v)[I(x,y) - I(x-u, y-v)]^2, \tag{3.1}$$

where (x,y) is the location of a pixel with color value $I(x,y)$, and $u,v \in 1,2,3$ model the move on each dimension. At a given point (x,y) of the image, the value of $S(x,y)$ is compared to a suitable *threshold* in order to determine the nature of the corresponding pixel. Roughly speaking, the process is achieved by applying four discrete operators, namely *Sobel* (S), *Multiplication* (M), *Gauss* (G), and *Coarsity* (C). Figure 3.10 displays an overview of the global workflow.

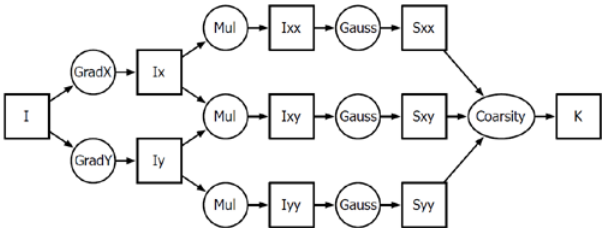


Figure 3.10: Harris algorithm diagram

Multiplication and *Coarsity* are point to point operators, while *Sobel* and *Gauss*, which approximate the first and second derivatives, are $9 \to 1$ or 3×3 operators defined by

$$S_x = \frac{1}{8}\begin{pmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{pmatrix} \quad S_y = \tfrac{1}{8}\begin{pmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{pmatrix} \tag{3.2}$$

$$G = \frac{1}{16}\begin{pmatrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{pmatrix} \tag{3.3}$$

Applying a 3×3 operator to a given pixel (x, y) consists in a point-to-point multiplication of the corresponding 3×3 matrix by the following pixels matrix

$$\begin{pmatrix} I(x-1,y-1) & I(x-1,y) & I(x-1,y+1) \\ I(x,y-1) & I(x,y) & I(x,y+1) \\ I(x+1,y-1) & I(x+1,y) & I(x+1,y+1) \end{pmatrix} \tag{3.4}$$

Here comes the notion of *border*. In order to compute the output pixel $O(x, y)$, we need the pixel $I(x, y)$ and those of its neighborhood. We say the operator is of *depth* 1. Operator depth is additive, means that if two operators f and g are of depth p and q respectively, then the depth of $f \circ g$ is $p + q$. Three problems are raised by the way operators are applied:

- Accessing the points at the periphery yields an irregular memory access pattern, which is a serious performance issue because of the sever penalty of cache misses.

- Computing two consecutive points involves some reused pixels (those on their common border). This yields redundant memory accesses and computation, thus another performance issue.

- Applying each convolution kernel separately implies several read/write operations on/to main memory (same location or not), yet another source of performance penalty.

There are several ways to deal with the above problems. One way is to fuse or chain consecutive operators whenever possible. This overcome the repetitive read/write of the entire image, at the price of data and computation redundancy (more border pixels), thus should be done under a certain compromise. The first two issues are well tackled by tiling, which could be considered with fused operators. Although tiling is a more general technique, we really need a specific analysis in order to understand how the extra data that covers each incoming tile affect the global performance when dealing with operator-based algorithms.

3.4.4 Experimental results

The goal here is to validate our implementation over various tile shapes, and see how close we are to our prediction of the optimal tile shape. The main program is executed from the PPE, which orchestrates the work of the cooperating PEs. For each image, we chose a fixed tile volume and then iterate on various shapes.

$tile_h$	$tile_w$	total time(s)
8	512	0.0494
16	256	0.0598
32	128	0.0485
64	64	0.0345
128	32	0.0517
256	16	0.0699
512	8	0.0734

Table 3.1: 512× 512 image

$tile_h$	$tile_w$	total time(s)
8	512	0.198
16	256	0.238
32	128	0.187
64	64	0.110
128	32	0.180
256	16	0.218
512	8	0.352

Table 3.2: 2048× 512 image

$tile_h$	$tile_w$	total time(s)
5	1200	0.494
10	600	0.360
20	300	0.264
40	150	0.235
80	75	0.183
160	37	0.247
320	18	0.275

Table 3.3: 1200× 1200 image

$tile_h$	$tile_w$	total time(s)
8	512	0.985
16	256	0.726
32	128	0.643
64	64	0.438
128	32	0.692
256	16	0.866
512	8	1.422

Table 3.4: 2048× 2048 image

We see that the most squared tile always gives the best global performance. The difference is marginal with closest shapes, but we should keep in mind that the typical use of the algorithm is with a flow of images. Our implementation does not overlap DMA with computation because of the necessary postprocessing due to memory misalignment. This aspect should be studied in the future, probably at the level of the DMA framework. For wider images (Tables 3.3 and 3.4), we see that the improvement using a square tile is more than 50% compared to the full row tile, which should be easier and fastest since it involves contiguous blocks memory. We emphasize on the extra cost for managing irregular DMAs, although our implementation seems to perform well. The main difference between full row tiles and rectangular tiles is that, for the later, a list DMA is always necessary. Thus, the compromise here is between irregular DMAs and redundancies. Our experimental results clearly show that it still interesting to consider tiles with balanced dimensions.

3.5 The algebraic path problem

3.5.1 abstract

The *algebraic path problem* (APP) unifies a number of related combinatorial or numerical problems into one that can be resolved by a generic algorithmic schema. In this work, we propose a linear SPMD model based on the *Warshall-Floyd* procedure coupled with a systematic shift-toroïdal. Our scheduling initially requires n processors for a $n \times n$ matrix to achieve the $O(n^3)$ task in $n^2 + O(n)$ steps. However, with a fewer number of processors, $p < n$, we exploit the modularity revealed by our linear array to achieve the task in $n^3/p + O(n)$ after n/p rounds, using a *locally parallel and globally sequential* (LPGS) partitioning. In any case, we just need each processor to have a local memory large enough to house one (block) column of the matrix. These two characteristics clearly justify an implementation on the CELL Broadband Engine, because of the efficient and asynchronous SPE to SPE communication (for the pipeline) and the floating point performance of each SPE. We report our experimentations on a QS22 blade with different matrix sizes, thus exhibiting the efficiency and scalability of our implementation. We show that, with a highly optimized Warshall-Floyd kernel (in Assembly), we could get close to 80 GFLOPS in single precision with 8 SPEs, which represents 80% of the peak performance for the APP on the CELL.

3.5.2 Introduction

The *algebraic path problem* (APP) unifies a number of related problems (*transitive closure, shortest paths, Gauss-Jordan elimination*, to name a few) into a generic formulation. The problem itself has been extensively studied at the mathematic, algorithmic, and programming point of view on various technical contexts. Among existing algorithms, the dynamical programming procedure proposed by Floyd for the *shortest paths* [6] and by Warshall for the *transitive closure* [30] so far remain on the spotlight. Due to the wide range of (potential) applications for this problem, also used as an ingredient to solve other combinatorial problems, providing an efficient algorithm or program to solve the APP is crucial.

In this work, we consider an implementation on the CELL [17]. The CELL processor has proved to be quite efficient compared to traditional processors when it comes to regular computation. For a more general use, an important programming effort is required in order to achieve the expected performance. Apart from providing highly optimized SPE kernels, it is also important to derive a global scheduling in which all participating SPEs efficiently cooperate in order to achieve the global task (managed from the PPE). Most of the existing codes for the CELL are based on a master slaves model, where the SPEs get the data from the PPE, perform the computation, and send the result back to the main memory through direct memory accesses (DMAs). Such models suffer from lack of scalability, especially on memory intensive applications. Our solution for the APP is based on a linear SPMD algorithm, with quite interesting properties like *local control*, *global modularity*, *fault-tolerance*, and *work optimal performance*.

Some attempts to implement the transitive closure on the CELL can be found in the literature. Among them, we point out the works described in [15] (up to 50 GFLOPS) and in [28] (up to 78 GFLOPS in perspective). The two solutions are both based on a block partitioning of the basic Warshall-Floyd procedure together with ad-hoc memory optimization and efficient global synchronization. With such *master-slaves* models where all the SPEs compute the same step of the *Warshall-Floyd* procedure at a time, a special care is required for *data alignment* in addition to redundant data management (the pivot elements). Moreover, since the memory is a critical resource for the SPE, we think it is important to come with a solution which is less memory constrained. Our answer to this demand is a pipelined algorithm efficiently implemented on the CELL, which we find to be a valuable contribution, both from the methodology point of view and its competitive absolute performance (potential of 80 GFLOPS).

3.5.3 The algebraic path problem

Formulation

The *algebraic path problem*(APP) may be stated as follows. We are given a weighted graph $G = \langle V, E, w\rangle$ with vertices $V = \{1, 2, \cdots, n\}$, edges $E \subseteq V \times V$ and a weight function $w : E \to S$, where S is a *closed* semiring $\langle S, \oplus, \otimes, *, \mathbf{0}, \mathbf{1}\rangle$ (*closed* in the sense that $*$ is a unary "closure" operator, defined as the infinite sum $x* = x \oplus (x \otimes x) \oplus (x \otimes x \otimes x) \oplus \cdots$). A *path* in G is a (possibly infinite) sequence of nodes $p = v_1 \cdots v_k$, and the *weight* of a path is defined as the product $w(p) = w(v_1, v_2) \otimes w(v_2, v_3) \otimes \cdots \otimes w(v_{k-1}, v_k)$. Let $P(i, j)$ denotes the (possibly infinite) set of all paths from i to j. The APP is the problem of computing, for

all pairs (i, j), such that $0 < i, j \leq n$, the value $d(i, j)$ defined as follows

$$d(i,j) = \bigoplus_{p \in P(i,j)} w(p). \tag{3.5}$$

For the *transitive closure*, M is the incidence boolean matrix and $\{\oplus, \otimes\} = \{\vee, \wedge\}$. For the *shortest path*, M is the cost matrix and $\{\oplus, \otimes\} = \{\min, +\}$. In any case, $\oplus$ and $\otimes$ are *commutative* and *associative*. Moreover, $\otimes$ is distributive over $\oplus$. These three properties are very important as they allow to safely permute and factorize the computations as desired.

Warshall-Floyd algorithm

If M is the *incidence or weight* matrix of a finite graph G of order n, then $M^{(k)}$ denotes the matrix of $d^{(k)}(i, j)$ (distance between i and j i to j considering intermediate nodes from 1 to k), and M^* the closure matrix (the one we want to compute). By extending the operator $\oplus$ to matrices, we obtain

$$M^* = M^{(0)} \oplus M^{(1)} \oplus M^{(2)} \oplus \cdots \oplus M^{(n)}, \tag{3.6}$$

where $M^{(0)} = M$.

The *Warshall-Floyd* dynamical programming procedure to solve the APP formulation is inspired from equation (3.6). Thus, $m^{(k)}(i, j)$ can be computed from $m^{(k-1)}(i, j)$ by considering node k as follows

$$m_{ij}^{(k)} = m_{ij} \oplus (m_{ik}^{(k-1)} \otimes m_{kj}^{(k-1)}). \tag{3.7}$$

An important property of this algorithm, which turns to be a memory advantage, is that the successive $M^{(k)}$ can be housed inside the same matrix M on memory. So, we perform n *in-place* (matrix) updates within the input matrix M and end up with the closure matrix M^*. At step k, row k (resp. column k), called *pivot row* (resp. *pivot row*), is used to upgrade $M^{(k-1)}$ to $M^{(k)}$.

A part from the $O(n^3)$ floating point operations, it is important to notice that the move of the pivot row and the pivot column, although quite regular compared to *gaussian pivoting*, needs a special attention. There are mainly two impacts. The first one is on the memory access pattern, because the pivots are shifted between one step and the next one. The second one is on the pipeline scheduling, the pivot elements have to be ready before starting the corresponding Warshall-Floyd step. In order to get rid of the difference between Warshall-Floyd steps, we now consider a toroïdal shift proposed by Kung, Lo, and Lewis [11].

Kung-Lo-Lewis mapping

The idea is to maintain the pivots at the same location, preferably on the axes. To do so, Kung, Lo, and Lewis suggested a shift-toroïdal of the matrix after each application of the standard Warshall-Floyd procedure. Technically, this is equivalent to say that after each step, the nodes are renumbered so that node i becomes node $i - 1$ (or $(i - 1) \bmod n + 1$ to be precise). Thereby, the matrices $M^{(k)}$ become completely identical, with the pivot row (resp. pivot column) remaining the first row (resp. first column). There are two ways to handle such a reindexation. The first one is to explicitly shift the matrix after the standard Warshall-Floyd procedure. The second one is perform the shift-toroïdal on the fly, means after each update of the matrix entries. Figure 3.11 depicts the two possibilities.

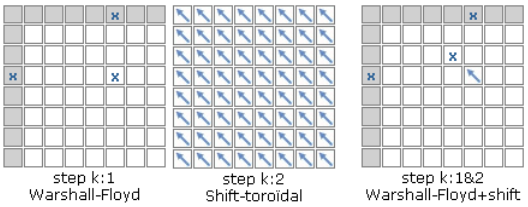


Figure 3.11: Toroïdal shift

In a formal point of view, we apply the following rule

$$m_{i-1,j-1}^{(k)} = m_{ij} \oplus (m_{ik}^{(k-1)} \otimes m_{kj}^{(k-1)}), \tag{3.8}$$

where operations on subscripts are performed modulo n. When implementing the algorithm in this way, one needs to keep away the pivot row and/or the pivot column (its depends on the scheduling) as they could be overwritten due to the shift. In a parallel context, where the data move between the computing units, the aforementioned shifts could be done by just adapting the transfer patterns accordingly (i.e. *data transfers + shifts* are thus performed at the same time at the price of the transfers only). We take all these into account to derive our linear pipeline scheduling.

3.5.4 Description of our algorithm

Scheduling

Given a graph of order n, our scheduling can be intuitively described as follows. The computation of $M^{(k)}$, assigned to a single processor, is performed row by row, from the first row (the pivot) to the last one. Each row is computed from the first point (the pivot) to the last one.

If (i, j, k) refers to the (i, j) entry of $M^{(k)}$, then our scheduling can be expressed by the *timing* function t and the *task allocation* function a given by

$$t(i, j, k) = L(k, n) + (i \times n + j) + 1 \tag{3.9}$$

$$a(i, j, k) = k \tag{3.10}$$

where $L(k, n)$ is the *computation latency* from graph dependencies and the row-wise scheduling. At this point, we need n processors that cooperate on a linear basis. Each processor operates as follows:

- ◇ *compute* the first row (the pivot) and *keep* it on the local memory
- ◇ *compute and send* each of the remaining rows
- ◇ *send* the pivot row

Computing the pivot row requires n steps, which count for the computation latency as no value is sent out during that time. In addition, because of the rotation, a given processor computes a row in the order $0, 1, 2, \cdots, n-1$ and outputs the results in the order $1, 2, \cdots, n-1, 0$. Thus, the total latency between two consecutive processor is $(n+1)$, and we thus obtain

$$L(k, n) = (k-1)(n+1), k \geq 1. \tag{3.11}$$

So, processor k starts at step $L(k,n) = k(n+1)$ and ends its task n^2 steps after (i.e. at step $n^2 + k(n+1)$). It is important to keep these two values in mind as they will be locally used by each processor to asynchronously distinguish between computing phases. Our solution originally needs n processors, which is a strong requirement in practice. Fortunately, the conceptual modularity of our scheduling naturally helps to overcome the problem as we now describe.

Modularity

Recall that processor k computes $M^{(k)}$ and communicates with processors $k-1$ and processor $k+1$. If we have p processors, $p < n$, then we adapt our schedule by just requesting processor k to computes $M^{(k+\alpha p)}$, for all integers α such that $k + \alpha p \leq n$. This is naturally achieved by performing several rounds (n/p to be precise) over our linear array of p processors. This corresponds to the so-called *locally parallel and globally sequential* (LPGS). The fact that our steps are completely identical makes it really natural to implement. Moreover, there is no additional memory requirement. Indeed, the capability of performing all updates within the same matrix is still valid, processor 0 continuously reads from A and processor $p-1$ continuously writes to A (there will be no read/write conflict since they always act on disjoint memory locations).

The remaining part of $M^{(\alpha p)}$ and the yet computed part of $M^{((\alpha+1)p)}$ will reside in the same matrix space into the main memory. Moreover, the idempotent property of the APP (i.e. $M^{(n+k)} = M^{(n)} = M^*, \forall k \geq 0$) provides another simplicity. Indeed, if p does not divides n, then a strict application of our partitioning will end up with $M^{(m)}$, where $m = \lceil (n/p) \rceil \times p$ is greater that n. We will still get the correct result, but with an additional $p - (n \bmod p)$ steps. If we do not want this additional unnecessary computation, we could just dynamically set processor $n \bmod p$ to be the last processor at the beginning of the ultimate round.

Because of the communication latency, it is always faster to perform block transfers instead of atomic ones. From a starting fine-grained scheduling, this is achieved by tiling. Since we plan to implement our algorithm at row level (i.e. we compute/send rows instead of single entries), applying a standard tiling just means globally block row partitioning and locally block column partitioning.

Tiling

Considering the original APP formulation and the *Warshall-Floyd* algorithm as previously described, a tile version can be derived by extending the atomic operations $\oplus$ and $\otimes$ to operate on blocks. Now, each step is either the resolution of the APP on a $b \times b$ subgraph, or a "multiply-accumulate" operation $A \oplus (B \otimes C)$, where the operands are $b \times b$ matrices and the operations are the matrix extension of the semiring operators. The only structural change imposed by a block version is that the row pivot (resp. column pivot) need to be explicitly updated too (they do not remain constant as in the atomic formulation). An important question raised by the tile implementation of our linear algorithm is the optimal tile size. Indeed, tiling yields a benefit from the communication latency, but at the price of the global computation latency (i.e. $p \times n$, which now becomes $p \times (bn)$). We now explain our implementation on the CELL. The goal is to validate our algorithm and discuss about *latency, tiling* and *scalability*.

3.5.5 Performance results

All our experimentations are performed on a QS22 CELL Blade with single precision data. First, we need to see how tiling affects the performance of our program. In table 3.5, we use our algorithm to solve the APP on a single SPE with matrices of size 128×128, 256×256, and 512×512 respectively (times are in seconds).

Tile	128×128	256×256	512×512
1	0.0216	0.1321	0.8921
4	0.0110	0.0823	0.6371
8	0.0104	0.0782	0.6082
12	0.0092	0.0754	0.5839
16	0.0105	0.0781	0.6017
20	0.0095	0.0696	0.5757
24	0.0098	0.0704	0.5872
28	0.0088	0.0782	0.5901
32	0.0115	0.0815	0.6119

Table 3.5: Relative impact of tiling

Recall that tile size b means we operate on the whole matrix by $b \times n$ block rows. What we see is that, a part from the fine grained computation, the variance using different tile sizes is marginal. This is certainly due the fact that the matrix operations dominate as we will see on the next results. However, as previously mentioned, our kernel for the $b \times b$ APP is not sufficiently optimized, otherwise we would have certainly observed a more significant performance gap. Nevertheless, we observe a factor 2 improvement between using tile of size 20 and the fine-grained version for instance. Now, we reconsider the same matrices and perform a scalability test from 1 SPE to 8 SPEs. In figure 3.6, we display the global execution time (measured from the PPE) with various tile sizes in $\{1, 4, 8, 12, 16\}$ (we stop at 16 because 12 seems to be the optimal), σ refers to the speedup compared to 1 SPE.

	1 SPE	2 SPEs		8 SPEs	
Tile	t(s)	t(s)	σ	t(s)	σ
1	1.3	1.25	1.06	0.31	4.29
4	0.8	0.41	2.00	0.11	7.78
8	0.7	0.39	1.99	0.11	7.21
12	0.7	0.36	2.08	0.10	7.93
16	0.7	0.40	1.97	0.14	5.65

(a) Performance with a 256×256 matrix

	1 SPE	2 SPEs		8 SPEs	
Tile	t(s)	t(s)	σ	t(s)	σ
1	0.892	0.434	2.05	0.213	4.19
4	0.637	0.318	2.00	0.080	7.96
8	0.608	0.304	2.00	0.078	7.79
12	0.584	0.293	1.99	0.074	7.88
16	0.602	0.302	1.99	0.083	7.23

(b) Performance with a 512×512 matrix

	1 SPE	2 SPEs		8 SPEs	
Tile	t(s)	t(s)	σ	t(s)	σ
1	6.67	3.28	2.03	1.60	4.16
4	5.01	2.50	2.00	0.62	7.99
8	4.79	2.39	2.00	0.60	7.95
12	4.70	2.32	2.02	0.58	7.98
16	4.72	2.36	2.00	0.60	7.79

(c) Performance with a 1024×1024 matrix

Table 3.6: Timings on a CELL QS22

Apart of the fine-grained version (first row of the results), we observe a perfect scaling

of our program. In order to illustrate the efficiency of our method (scheduling + DMA + synchronization), we show in table 3.7 the timings using a full version of our program where the block APP kernel is not executed. We clearly see that the overhead due to our pipeline mechanism is definitely negligible, thus the overall performance just relies on the block APP kernel. By replacing our APP kernel code by a fast implementation similar to that of the matrix product in [35], our implementation achieves 80 GFLOPS (note that the peak performance on the APP is 102 GFLOPS as the "multiply-add" cannot be used).

	1 SPE	2 SPEs		8 SPEs	
Tile	t(s)	t(s)	σ	t(s)	σ
1	1.87	1.20	1.56	0.53	3.49
4	0.39	0.47	0.83	0.11	3.70
8	0.19	0.12	1.64	0.06	3.78
12	0.12	0.08	1.65	0.03	4.02
16	0.09	0.06	1.63	0.03	3.78

Table 3.7: DMA timings (1024×1024 matrix)

rr

3.6 Lattice Quantum Chromodynamics library

3.6.1 Abstract

Quantum chromodynamics (QCD) is the theory of subnuclear physics, aiming at modeling the strong nuclear force, which is responsible of the interactions between nuclear particles. Numerical QCD studies are performed through a discrete analytical formalism called LQCD (Lattice Quantum Chromodynamics), from which numerical simulations are performed. LQCD simulations involve very large volume of data and numerically sensitive entities, thus the crucial need of high performance computing systems. The most heavy calculation requires to solve a huge and numerically sensitive linear system. For this purpose, iterative methods are definitely considered. Therefore, the corresponding matrix-vector product, also so-called *Wilson-Dirac operator*, appears as a critical computation kernel. This work was mainly motivated by the aim of providing an efficient accelerated implementation of the *Wilson-Dirac operator* an associated linear algebra subroutines on the CELL B.E.. Our framework is provided as a unified library and is particularly optimized for an iterative use. Each routine is parallelized among the SPEs, and each SPE achieves it task by iterating on the entire array in main memory by small chunks. The SPE code is vectorized with double precision data and we overlap memory accesses and computations. Moreover, we permanently keep the SPE context alive and we use mailboxes to synchronize between consecutive calls. We validate our library by using it to derive a CELL version of an existing LQCD package (tmLQCD). Experimental results on each routine in double precision show a significant speedup compare to standard processors, 11 times better than a 2.83 GHz INTEL quad-core processor for instance (without SSE, but multi-threaded). This ratio is around 9 (with QS22 blade) for the full *Wilson-Dirac inversion.*

3.6.2 Introduction

Quantum chromodynamics (QCD) [31], the theory of the strong nuclear force, can be numerically simulated on massively parallel supercomputers using the method of lattice gauge

theory (LQCD), see Vranas et al [29]. A LQCD simulation chain involves basic linear algebra computations on large scale entries. Moreover, basic operations are repeated so many times following a stopping criterion, which is either purely numerical or based on the physical meaning of the result. One major kernel is the inversion of the *Dirac operator*, which is an important step during the synthesis of a statistical gauge configuration sample. Indeed, in the Hybrid Monte Carlo (HMC) algorithm [26], it appears in the expression of the *fermionic force*, used to update the momenta associated with the gauge fields along a trajectory.

A common way to parallelized LQCD applications is to partition the lattice into sublattices and then assign each sublattice to a computing node (see [4, 14]). This yields a standard SPMD model which is then mapped onto a given parallel machine. Thus, tuning an individual computing node to efficiently perform a critical part of the simulation is a good way towards a powerful LQCD supercomputer. Number of authors have studied LQCD implementation on various supercomputers [29]. For the special case of solving the *Wilson-Dirac system*, a mixed-precision solution accelerated with GPUs is proposed by Clark[4]. A domain decomposition approach associated with the deflation technique is studied by Luscher[14]. A prospective overview of QCD implementation on the CELL is reported in [1]. A specific study of the *Dirac operator* (the most CPU consuming kernel) on the CELL (simulator) is reported in [8].

LQCD computation kernels are built up from basic linear algebra routines with special data structures. Hence, comes the idea of a computing library dedicated to LQCD. A well known example of such a library is the so-called QDP++[19], which provides a data-parallel programming environment suitable for Lattice QCD. In this work, we propose a CELL-accelerated library for the same purpose. Our main concern is performance. Indeed, having a more powerful node yields the additional advantage of a smaller computing network request, which significantly reduce the inter-nodes communication overhead. This aspect is crucial in LQCD performance, otherwise people consider very large supercomputers and suffer from communication penalty, and of course a high running cost.

3.6.3 Background and preliminaries

Foundations

Definition 3 *Given a complex square matrix A and an integer μ, we state the followings definitions*

- $\diamond$ $|A|$ *denotes the order of matrix* A
- $\diamond$ I_A *is the identity matrix of order* $|A|$
- $\diamond$ $A^\dagger = \bar{A}^T$
- $\diamond$ $\hat{\mu} = e_\mu$ *(μ^{th} vector of the canonical basis)*

Definition 4 *Given two matrices A and B, the tensor product $C = A \otimes B$ is defined by*

$$C = (a_{ij}B) \tag{3.12}$$

Property 1 *Given three complex matrices A, B and C, we have*
Associativity:

$$A \otimes (B \otimes C) = (A \otimes B) \otimes C \tag{3.13}$$

Normal factors decomposition:

$$A \otimes B = (A \otimes I_B)(I_A \otimes B) = (I_A \otimes B)(A \otimes I_B) \tag{3.14}$$

Tensor product is rarely computed explicitly as it involves a huge amount of memory and lot of computation redundancies. Instead, an implicit approach is commonly considered depending on the desired operation [5].
We now consider five 4×4 special matrices, called *Dirac γ-matrices*, which are given by

$$\gamma_0 = \begin{pmatrix} 0 & 0 & -1 & 0 \\ 0 & 0 & 0 & -1 \\ -1 & 0 & 0 & 0 \\ 0 & -1 & 0 & 0 \end{pmatrix} \quad \gamma_1 = \begin{pmatrix} 0 & 0 & 0 & -i \\ 0 & 0 & -i & 0 \\ 0 & i & 0 & 0 \\ i & 0 & 0 & 0 \end{pmatrix} \tag{3.15}$$

$$\gamma_2 = \begin{pmatrix} 0 & 0 & 0 & -1 \\ 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \\ -1 & 0 & 0 & 0 \end{pmatrix} \quad \gamma_3 = \begin{pmatrix} 0 & 0 & -i & 0 \\ 0 & 0 & 0 & i \\ i & 0 & 0 & 0 \\ 0 & i & 0 & 0 \end{pmatrix} \tag{3.16}$$

$$\gamma_5 = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & -1 & 0 \\ 0 & 0 & 0 & -1 \end{pmatrix} \tag{3.17}$$

The *Wilson-Dirac operator* the following generic form:

$$\begin{aligned} D\psi(x) = A\psi(x) \; - \\ \frac{1}{2}\sum_{\mu=0}^{4}\{ \; & [(I_4 - \gamma_\mu) \otimes U_{x,\mu}]\psi(x + \hat{\mu}) + \\ & [(I_4 + \gamma_\mu) \otimes U^{\dagger}_{x-\hat{\mu},\mu}]\psi(x - \hat{\mu})\} \end{aligned} \tag{3.18}$$

where

- A is a 12×12 complex matrix of the form $\alpha I_{12} + \beta(\nu \otimes \gamma_5)$, where α, β are complex coefficients and ν a 3×3 complex matrix
- x is a given point of the lattice (a *site*), which is a finite subset of $\mathbb{N}^4$
- ψ (called *quark field* or *Wilson vector*) is a 12-components complex vectors
- $U_{x,\mu}$ is a 3×3 complex matrix (called *gluon field matrix* or *gauge matrix*) at (x, μ).

At a given lattice point x, $\psi(x)$ (called *quark spinor* or *spin-color vector*) is a 12-components complex vector. For a given *quark field* ψ, $D\psi$ is obtained by considering $D\psi(x)$ for all points within the lattice, and the result is a vector of the same size as for the input. In addition, for a given x, $D\psi(x)$ is a linear combination of the components of $\psi(x)$. Thus, it is consistent to see $D\psi$ as a matrix-vector product, and thereby consider D as an implicit square matrix (commonly referred to as the *Wilson-Dirac matrix*). The corresponding operation (3.18) is the most time consuming kernel as it involves a significant amount of floating point operations on larger lattices and is done very frequently. However, other linear algebra operations are

also time consuming because they are applied on very large arrays of complex numbers. One advantage when using the CELL on such memory intensive processing is the possibility of overlapping data transfers and computations, similarly to the classical memory prefetch concept on standard processors. We describe the basic data structures commonly used in LQCD calculations.

Data structures

The aspect of data structure is important in the LQCD community, as it concerns the interoperability of existing software packages and files format. Typical data structures used in LQCD are based on the following data types.

```
typedef struct
{
  double re,im;
} complex;

typedef struct
{
  complex c00,c01,c02,c10,c11,c12,c20,c21,c22;
} su3;

typedef struct
{
  complex c0,c1,c2;
} su3_vector;

typedef struct
{
  su3_vector s0,s1,s2,s3;
} spinor;

typedef struct
{
  su3_vector s0, s1;
} halfspinor;
```

Note that any arrays based on one of the above data structures can be manipulated as an array of contiguous double precision numbers (i.e. pointer on `double`). Table 3.8 provides the aforementioned equivalences.

Original type	Equivalent type	size (bytes)
`complex c;`	`double c[2];`	16
`su3 u;`	`double u[18];`	144
`su3_vector v;`	`double v[6];`	48
`spinor s;`	`double s[24];`	192
`halfspinor h;`	`double h[12];`	96

Table 3.8: Data structures equivalence

The inputs/outputs of the routines implemented in our library are of type `double`, `complex`, or `spinor`. At runtime, DMAs operate on arrays of `spinor`. Thus, since a `spinor` has size 192 = 16×12 bytes, aligning the entry pointer of the whole array of spinors is sufficient to have the DMA of any subarray being 16 bytes aligned. In the case of the *Wilson-Dirac* operator, we also transfer `su3` data (U matrices) together with the `unsigned int` data representing the indices of the right hand side spinors (see equation (3.18)). From equation (3.18), we also see that eight U matrices and eight indices are needed for each spinor component. Thus, we transfer 8K `su3` and 8K `unsigned int`, which are both of a suitable size for a DMA (multiple of 16 bytes). Let now see how each of our CELL-accelerated routines are built.

3.6.4 Generic acceleration scheme

For each of the selected routines, we perform the acceleration on the CELL through three main mechanisms.

Single Instruction Multiple Data

We derive a SIMD version of the code using the SPE intrinsics provided by IBM. With double precision data, each vector register (16 bytes length) can handle two values. An illustrative example is provided through Procedure 1 which subtracts two `su3_vector` (array of 3 complex numbers).

Procedure 1 *SIMD subtraction of two* `su3_vector`

```
1: _SPU_vector_add(su3_vector *r, su3_vector *s1, su3_vector *s2)
2: v_1 = (vector double *)&(s1);
3: v_2 = (vector double *)&(s2);
4: v_3 = (vector double *)&(r);
5: v_3[0] = spu_sub(v_1[0], v_2[0]);
6: v_3[1] = spu_sub(v_1[1], v_2[1]);
7: v_3[2] = spu_sub(v_1[2], v_2[2]);
```

For some cases like those involving the product of two complex numbers, register swapping instructions are necessary. This additional processing has a cost, thus preventing a perfect SIMD scaling. Unfortunately, this cannot be avoided for free when processing with complex numbers. However, the benefit of the vectorization is still noticeable, especially if swapping instructions and calculation instructions are soundly pipelined.

Task partitioning

We mainly follow the standard manager/workers scheme as it commonly applied when scheduling the tasks among the SPEs of the CELL. The PPE is the host and the manager of the whole task. For each subroutine to be accelerated, the task is block partitioned, and each portion is assigned to one SPE through the corresponding thread. Then, each SPE iterates by chunks over its own subregion while performing the required calculation. This is semantically equivalent to a loop partitioning followed by a inner loop blocking. Figure fig. 3.12 provides an overview of the workflow.

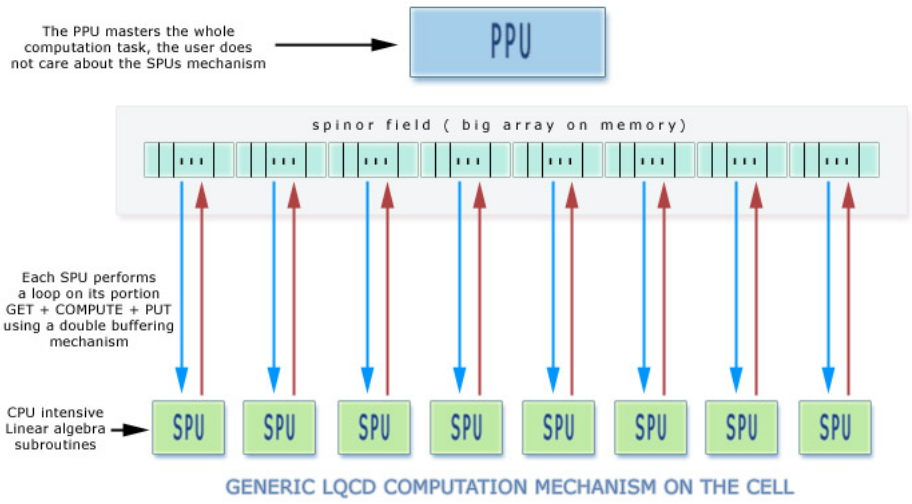


Figure 3.12: Global SPUs-acceleration mechanism

The SPE executes its task from a data partitioning of its domain. Once it has received the parameters of its iteration subspace, the generic computation is repeatedly performed on each chunk of arrays (the maximum that fits into the local store) until the whole assigned domain is covered. The typical iteration follows the scheme `DMA_get + SIMD Computation + DMA_put`. Since we need to perform two DMAs per step (one for getting the block data and another one for sending the result back to the main memory), double buffering is implemented by interleaving and overlapping memory accesses with computation (another benefit of splitting the computation) as we now explain.

DMA and Double buffering

When it comes to DMA on LQCD calculations, there are several issues to overcome. The first one concerns the alignment of the data. As we already explained, the case of `spinor` arrays is rather simple because each `spinor` has a size of 192 bytes, which is a multiple of 16 bytes. For `su3` data, the situation needs a little more attention, since each `su3` data (a U matrix) has a size of 144 bytes, which is not a multiple of 16 bytes as required for a DMA. This problem is sometimes solved by aggregating the height required `su3` data for each `spinor` into a contiguous block memory. The second DMA issue concerns the size of the data, even for a single computation. Indeed, for the *Wilson-Dirac* operator (see (3.18)), calculating a single *spinor* requires

- ◇ 8 *spinors*,
- ◇ 8 *SU(3) matrices*,
- ◇ 8 indexes (pointer to the 8 neighbor *spinors* to be used)

Thus a total of $8 \times 192 + 8 \times 144 + 8 \times 8 = 2752$ bytes. Therefore, if we want to compute a block of N spinors, keeping in mind that the size of the SPE local memory is 256 KB (262 144 Bytes), the maximum number of spinors we can compute in one step (namely N_{max}) is constrained by $2752 \times N_{max} \leq 262\ 144$, means $N_{max} \leq 95$. Taking into account the part

allocated for the program itself, we found $N = 64$ as the best working value. We see that the optimal working data volume for a single block computation on the SPE is fixed and does not depend on the lattice size, thus a good modularity of the global computation (bigger lattice just implies more iterations). Figure 3.13 provides the timings of our DMA implementation and the ratio over the total execution time for the *Wilson-Dirac* operator.

nb SPUs	time(μs)	% comp. time
1	0.56	24%
2	0.29	25%
3	0.22	27%
4	0.18	29%
5	0.21	37%
6	0.21	40%
7	0.16	37%
8	0.10	34%

Figure 3.13: DMA timings for the Dirac operator

From figure 3.13, we see that even for the heaviest kernel, DMAs take around 30% of the total computing time. This clearly shows that the DMAs cost can be perfectly hidden by overlapping them with double precision computation (both for QS20 and QS22 blades). For single precision computation (not yet tested at this time), we think we still have room for an acceptable partial overlapping. Procedure 2 gives a prototype of our computation/DMA relative scheduling.

Procedure 2 *Generic double buffering scheme*

```
1: DMA_get_request_data(0);
2: for (k=0; k < N; k++) do
3:    DMA_get_waitfor_data(k);
4:    DMA_get_request_data(k+1);
5:    SPU_compute_range(k, a, (a+b)/2);
6:    if (k>0) then
7:       DMA_put_waitfor_data(k-1);
8:    end if
9:    SPU_compute_range(k, (a+b)/2 + 1, b);
10:   DMA_put_request_data(k);
11: end for
12: DMA_put_waitfor_data(N-1);
```

The generic DMA mechanism of Procedure 2 considers two-ways transfers. We need to make sure that a partial result as been sent back to the main memory before we reuse the same space for the next computation. That's why we split the SPE computation into two parts. The first part should overlap with the DMA_put of the previous result, while the second one should overlap with the DMA_get of the next input. For reduction operations like the scalar product or the square norm, this additional strategy is not needed because we don't send any data during the whole computation (we just send the final result at the end). However, if we chose to keep the global mechanism for reduction operations too, then we will be able to do an assignment operation at the same time. Thus, we provide the routine `CELL_square_norm_assign(spinor *S, spinor *R, int N)` which calculates the square norm of the spinor arrays R while copying it into S. We then eliminate the cost of the memory copy by hiding it into the square norm

calculation. This local optimization, applied in many parts of our library, is not negligible as memory accesses is a critical part of LQCD computations.

Eliminating the overhead of threads creation

SPE threads are created and launched from the PPE in a sequential way. With several SPEs, the corresponding overhead becomes more important, especially if we create and destroy the threads at each iteration. Consequently, as the SPE code is improving (by code optimization, faster hardware, or single precision calculation), the overall cost of creation/destruction of the threads might become dominant. What we did was to keep each created SPE thread permanently alive during the entire computation. We use a generic master data structure for all our SPE routines. The corresponding variables, one for each SPE, contain scalar parameters (if any) and pointers to the input/output arrays. The SPE first initiates a DMA to get its own master structure, and then uses it to perform it task following the scheme of Procedure 2. The master structure remains the same (we only change its values) during different calls. By this way, each SPE thread is created once and kept active during a complete session with our library. The use of our library at the SPE level is orchestrated by the code of figure 3.14, combined with the full binary code. So, the SPE just need to select which routine to execute using an indicator sent by the PPE using the mailbox mechanism.

```
int main(int argp){
  unsigned int which_code;
  /* SPE asks what to do */
  which_code = spu_read_in_mbox();
  while(which_code!=-1){
     switch(which_code){
       case 0: routine_0(argp); break;
       case 1: routine_1(argp); break;
       ...
       case n: routine_n(argp); break;
       default: break;
     }
     /* SPE tells he has finished */
     spu_write_out_mbox(1);
     /* SPE waits for next run */
     which_code = spu_read_in_mbox();
  }
  return 0;
}
```

Figure 3.14: SPE main program

At the initialization of the library, the PPE creates and loads the context on each participating SPE. During a session with our library, when the user requests the execution of `PPE_routine_id`, the PPE prepares the data and mails `id` to the SPEs. Each SPE receives `id`, executes `SPE_routine_id`, and waits for the next request. Figure 3.15 displays the list of routines yet implemented in our library (see [10] for their original specifications).

```
void CELL_QCD_INIT();
void CELL_QCD_FINALIZE();
complex CELL_scalar_prod();
double CELL_scalar_prod_r();
double CELL_square_norm();
double CELL_square_norm_assign();
void CELL_assign_diff_mul();
void CELL_mul_r();
void CELL_assign();
void CELL_assign_mul_add_r();
void CELL_assign_diff_mul_serie();
void CELL_Hopping_Matrix();
void CELL_H_eo_tm_inv_psi();
void CELL_diff();
void CELL_mul_one_pm_imu_sub_mul_gamma5();
void CELL_mul_one_pm_imu_inv();
build_dependence_indices();
```

Figure 3.15: List of implemented routines

3.6.5 How to use the library

The library is intended to be integrated into any C (or C++) codes. The only one required change (if not yet done) concerns the memory allocations, which have to be 16 bytes aligned. Our suggestion is to use `#define` statements to change every `malloc` into `malloc_align` and `free` into `free_align`. This is done in the file `cell_lqcd.h`, which also contains the declaration of all the routines. The steps necessary to use the library are the followings:

- ◇ include the file `cell_lqcd.h` into the code
- ◇ compile the code and link it with the library
  `ppe_lqcd.o spu_lqcd.a -lspe2 -lmisc`

Typical use of the library follows the sequence below:

- ◇ `CELL_QCD_INIT();` (called once at the beginning)
- ◇ different calls to the routines
- ◇ `CELL_QCD_FINALIZE();` (called once at the end)

If `CELL_Hopping_Matrix()` needs to be used, then a call to `build_dependence_indices()` is necessary to build the dependence indices following formula (3.18). The result of this procedure is an array of indices that will be passed as an argument to `CELL_Hopping_Matrix()`. We chose to separate it from `CELL_Hopping_Matrix()` because it this done once.
The library can be dowloaded at
`www.omegacomputer.com/staff/tadonki/dirac/cellqcd.htm`

3.6.6 Performance results

We present some performance results using our library in a 32×16^3 lattice (hence 131 072 sites). We use 8 SPEs simultaneously.

	Intel	CELL BE	
	2.83Ghz	QS20	QS22
`cell_diff`	0.0110	0.00177	0.00183
`cell_scalar_prod`	0.0060	0.00148	0.00110
`cell_assign_diff_mul`	0.0080	0.00176	0.00175
`cell_Hopping_Matrix`	0.1210	0.01155	0.00650

Figure 3.16: Timings (seconds) of individual routines

We see from figure 3.16 that our implementation is globally competitive. The difference between QS20 and QS22 is perceptible only with the `Hopping_Matrix()` because of its significant computing load. This is in our favor, because `Hopping_Matrix()` implements the *Wilson-Dirac* operator, which consumes more that 80% of the computation time when dealing with linear system solving. Indeed, figure 3.17 displays the elapsed times for solving a *Wilson-Dirac* system using iterative methods (GCR: Generalized Conjuguate Residual and CG: Conjugate Gradient) on our 32×16^3 lattice.

	Intel	CELL BE	
	2.83Ghz	QS20	QS22
`GCR (57 iterations)`	27 s	5.58 s	3.68 s
`CG (685 iterations)`	362 s	42 s	20 s

Figure 3.17: Timings of the Wilson-Dirac inversion

Comparing our library executed the CELL with the original one executed on an Intel 2.83 Ghz (without SSE and using one core just to be close to the 3.2 GHz frequency of an SPE), we globally see a promising speedup (around 9). The case of CG is more impressive because the algorithm mainly relies on the *Wilson-Dirac* operator, which is the best accelerated routine of our library. With an improvement in the DMA organization, we got the following results on the Dirac operator.

	QS20			QS22		
#SPE	Time(s)	S	GFlops	Time(s)	S	GFlops
1	0.109	1.00	0.95	0.0374	1.00	2.76
2	0.054	2.00	1.92	0.0195	1.91	5.31
3	0.036	3.00	2.89	0.0134	2.79	7.76
4	0.027	3.99	3.85	0.0105	3.56	9.90
5	0.022	4.98	4.73	0.0090	4.15	11.56
6	0.018	5.96	5.78	0.0081	4.61	12.84
7	0.015	6.93	6.94	0.0076	4.92	13.88
8	0.013	7.88	8.01	0.0075	5.75	14.02

Figure 3.18: Dirac operator on the CELL

Without SSE		With SSE	
1 core	4 cores	1 core	4 cores
0.0820	0.0370	0.040	0.0280

Figure 3.19: Dirac operator Intel i7 quadcore 2.83 Ghz

We see from figure 3.18 that our implementation of the Dirac operator scales very well on a QS20 and suffers from a slowdown on a QS22. We think, at this stage of our work, that this is due to an inappropriate SPE allocation, since we use a dual Cell based blade. This should be easy to fix and then provide a scalable implementation on a QS22 too. Moreover, even with our optimal DMA organization, this part is still highly dominant on a QS22. The main idea we have in mind to overcome this is to use of the SU(3) reconstruct mechanism. This will significantly reduce the volume of exchanged data and increase the SPE computation load, thus a more balanced implementation.

3.6.7 Perspectives

The promising results of our work motivate to explore other sources of improvement. Among them, we might independently consider to:

- ◇ derive a **single precision version**. This will improve the SPE performance while reducing the DMA cost (volume + occurrences). The require adaptation looks straightforward at the SPE level, since we just need to change our `vector double` variables to `vector float`. However, the data layout consistency of the whole framework should be rechecked, as well as the numerical impact on the global iterative process (maybe could consider a mixed precision methods).

- ◇ **calculate the dependence indexes** directly on the SPEs instead of getting them from DMAs. At the price of an extra work for the SPE, the reward will be again a reduction of the DMA cost and a bigger space on the SPE local store to house more spinors (bigger TLP granularity).

- ◇ explore the impact of **reconstructing the U matrices** on the SPEs (from *12 numbers* parametrization or *8 numbers* parametrization) [4]. The same reward arguments as for the previous statement apply.

- ◇ generalized our CELL deployment methodology to other inversion paradigms (within the same package) like the Conjugate Gradient (CG).

- ◇ perform some experiments on a **cluster of CELL blades**. Since, a distributed memory parallelization through MPI is already implemented in the tmLQCD package [10], there will be any additional effort to run on a cluster of CELL blades, preferably with a high speed network like the Infiniband.

3.7 Conclusion and perspectives

Accelerated computing is a very promising way to provide efficient implementations for applications that has quite regular computation kernels. The price to achieve a high performance is the underlying programming efforts, which is really the critical point. In order to overcome

this technical barrier, the current trend is to consider code generation frameworks. From the global performance point of view, the problem of moving data from the host to the accelerators and vice-versa is still a noticeable limitation and should remain on the way for improvements. Another important point concerns the interprocessor communication when the computing nodes are accelerated. The question is how to exchange data between accelerators without going through the memory of the host processor. For the GPU, this topic is already under consideration. For numerical computation, reducing the performance slowdown when moving to double precision calculations is crucial to widespread the use of accelerators.

Bibliography

[1] F. Belletti, G. Bilardi, M. Drochner, N. Eicker, Z. Fodor, D. Hierl, H. Kaldass, T. Lippert, T. Maurer, N. Meyer, A. Nobile, D. Pleiter, A. Schaefer, F. Schifano, H. Simma, S. Solbrig, T. Streuer, R. Tripiccione, and T. Wettig. *QCD on the Cell Broadband Engine*, Oct 2007.

[2] D. Cachera, S. Rajopadhye, T. Risset, C. Tadonki, *Parallelization of the Algebraic Path Problem on Linear SIMD/SPMD Arrays*, IRISA report n 1409, july 2001.

[3] Cell SDK 3.0. `www.ibm.com/developerworks/power/cell`.

[4] M. A. Clark, R. Babichc, K. Barrose, R. C. Browerc, C. Rebbic *Solving Lattice QCD systems of equations using mixed precision solvers on GPUs*, `http://arxiv.org/abs/0911.3191`, 2009.

[5] Claude Tadonki and Bernard Philippe, *Parallel multiplication of a vector by a Kronecker product of matrices (part II)*, Parallel Distributed Computing Practices PDCP, volume 3(3), 2000.

[6] R. N. Floyd, *Algorithm 97 (shortest path)*, Comm. ACM, 5(6):345, 1962.

[7] C. Harris and M. Stephens, *A combined corner and edge detector*, 4^{th} ALVEY Vision Conference, 1988.

[8] K. Z. Ibrahim and F. Bodin, *Implementing Wilson-Dirac operator on the cell broadband engine*, ICS '08: Proceedings of the 22nd annual international conference on Supercomputing, pp. 4-14, Island of Kos, Greece, 2008.

[9] F. Irigoin and R. Triolet, *Supernode partitioning*, in 15th ACM Symposium on Principles of Programming Languages, pp. 319-328, ACM, january 1988.

[10] K. Jansen and C. Urbach, *tmLQCD: a program suite to simulate Wilson Twisted mass Lattice QCD*, Computer Physics Communications, vol. 180(12), p. 2717-2738, 2009.

[11] S.Y. Kung, S. C. LO, P. S. Lewis, *Optimal systolic design for the transitive closure and the shortest path problems*, IEEE transactions on computers ISSN 0018-9340, vol. 36(5), pp. 603-614, 1987.

[12] J. Kurzak and Jack Dongarra, *QR factorization for the Cell Broadband Engine*, Scientific Programming, vol. 17(1-2), P. 31-42, 2009.

[13] J. Kurzak, Alfredo Buttari, and Jack Dongarra, *Solving Systems of Linear Equations on the CELL Processor Using Cholesky Factorization*, `www.netlib.org/lapack/lawnspdf/lawn184.pdf`

[14] M. Luscher, *Implementation of the lattice Dirac operator*, 2006.

[15] K. Mastumoto and Stanislav G. Sedukhin, *A Solution of the All-Pairs Shortest Paths Problem on the Cell Broadband Engine Processor*, IEICE Trans. Inf. & Syst., Vol. E92.D, No. 6, pp.1225-1231, 2009 .

[16] H. Moravec, *Obstacle avoidance and navigation in the real world by a seeing robot rover*, In tech. report CMU-RI-TR-80-03, Robotics Institute, Carnegie Mellon University & doctoral dissertation, Stanford University. September 1980.

[17] H. Peter Hofstee, *Power Efficient Processor Design and the Cell Processor*, `http://www.hpcaconf.org/hpca11/slides/Cell_Public_Hofstee.pdf`.

[18] D. Pleiter, *QPACE: Power-efficient parallel architecture based on IBM PowerXCell 8i* ,EnA-HPC, Hamburg,17 September 2010.

[19] QDP++,
`http://usqcd.jlab.org/usqcd-docs/qdp++/`.

[20] T. Saidani, L. Lacassagne, J. Falcou, C. Tadonki, and S. Bouaziz, *Parallelization Schemes for Memory Optimization on the Cell Processor: A Case Study on the Harris Corner Detector*, HIPEAC Journal, 2009.

[21] S. Sen and S. Chatterjee, *Towards a theory of cache-efficient algorithms*, SODA 2000.

[22] C. Tadonki, *Large Scale Kronecker Product on Supercomputers*, 2nd Workshop on Architecture and Multi-Core Applications (WAMCA 2011) in conjunction with the International Symposium on Computer Architecture and High Performance Computing (SBAC PAD 2011), Vitria, Esprito Santo, Brazil, October 26-29, 2011.

[23] C. Tadonki, G. Grosdidier, and O. Pene, *An efficient CELL library for Lattice Quantum Chromodynamics*, International Workshop on Highly Efficient Accelerators and Reconfigurable Technologies (HEART) in conjunction with the 24th ACM International Conference on Supercomputing (ICS), pp. 67-71, Epochal Tsukuba, Tsukuba, Japan, June 1-4, 2010.
ACM SIGARCH Computer Architecture News, vol 38(4) 2011.

[24] C. Tadonki, L. Lacassagne, E. Dadi, M. Daoudi *Accelerator-based implementation of the Harris algorithm*, 5th International Conference on Image Processing (ICISP 2012), Agadir, Morocco, June 28-30, 2012.

[25] C. Tadonki, *Ring pipelined algorithm for the algebraic path problem on the CELL Broadband Engine*, Workshop on Applications for Multi and Many Core Architectures (WAMMCA 2010) in conjunction with the International Symposium on Computer Architecture and High Performance Computing (SBAC PAD 2010), Petropolis, Rio de Janeiro, Brazil, October 27-30, 2010.

[26] C. Urbach, K. Jansen, A. Shindler, and U. Wenger, *HMC Algorithm with Multiple Time Scale Intergration and Mass Preconditioning*, Computer Physics Communications, vol. 174, p. 87, 2006.

[27] G. Venkataraman, S. Sahni, and S. Mukhopadhyaya, *A blocked all-pairs shortest-paths algorithm*, J. Experimental Algorithmics, vol.8, no.2.2, 2003.

[28] S. Vinjamuri, V. K. Prasanna, *Transitive closure on the cell broadband engine: A study on self-scheduling in a multicore processor*, IEEE International Parallel & Distributed Processing Symposium (IPDPS), p. 1 - 11, Rome, Italy, 23-29 May 2009.

[29] P. Vranas, M. A. Blumrich, D. Chen, A. Gara, M. E. Giampapa, P. Heidelberger, V. Salapura, J. C. Sexton, R. Soltz, G. Bhanot, *Massively parallel quantum chromodynamics*, IBM J. RES. & DEV. VOL. 52 NO. 1/2 JANUARY/MARCH 2008.

[30] S. Warshall, *A Theorem on Boolean Matrices*, JACM, 9(1):11-12, 1962.

[31] F. Wilczek, *What QCD Tells Us About Nature and Why We Should Listen*, Nuc. Phys. A 663, 320, 2000.

[32] S. Williams, J. Shalf, L. Oliker, S. Kamil, P. Husbands, and K. Yelick, *Scientific Computing Kernels on the Cell Processor*, International Journal of Parallel Programming, 2007.

[33] S. Williams, J. Shalf, L. Oliker, S. Kamil, P. Husbands, and K. Yelick, *The potential of the Cell processor for scientific computing*, CF '06: Proc. 3rd Conference on Computing Frontiers, pp.9-20, ACM, New York, NY, USA, 2006.

[34] J. Xue, *Loop tiling for parallelism*, Kluwer 2000.

[35] `http://www.tu-dresden.de/zih/cell/matmul`

Chapter 4

Power aware computing

4.1 Abstract

This chapter raises of energy consumption issue from computing activities. There are mainly two contexts where the energy is one of the top priority concerns: *embedded computing* and *supercomputing*. For *embedded computing*, power consumption is critical because the amount of energy that is available for the devices is limited. For supercomputers, the heat dissipated is a serious source of failure, especially in the context of high-throughput computing. Cooling is the typical way to manage the issue, but its cost is likely to be a significant part of the maintenance budget. On a single computer, the problem is commonly considered through the electrical power consumption. This chapter, we discuss the problem and describe different formulations. Our formal analysis is articulated around our main contribution on the topic [28].

4.2 Overview of the energy concern and optimization

Due to the growing popularity of embedded systems [18, 19, 20], energy has emerged as a new optimization metric for system design. As the power availability in most of these systems is limited by the battery power of the device, it is critical to reduce energy dissipation in these systems to maximize their operation cycle. Power limitation is also motivated by heat or noise limitations, depending on the target application. For a multiprocessor system, the intuitive way is to optimize the energy consumption at the processor level (individually), assuming that the part coming from network activities can be neglected. Although this approach focuses on a local optimization, it can lead to a global scheduling strategy designed accordingly. The objective could be, for instance, to efficiently transition to the low-power states of each processor. In a low-power state, the processor is not active and has a reduced consumption, but it takes a transition delay to come back to an active state, thus a good compromise should be found. On a distributed memory machine, one could choose, for instance, to switch to an idle state on each blocking MPI communication, especially when waiting for data from another processor. In any case, a good profiling of the program to optimize is crucial to schedule the state transitions efficiently. This can be done from a static point of view following a good performance prediction model, or dynamically based on a suitable performance monitoring.

Cloud computing is also an important area where energy is an important concern. Indeed, computing and storage devices are continuously requested by different users. Such intensive use of resources implies a significant power consumption at various levels. One way to address the problem is through the concept of *federated clouds*, where different clouds are virtually merged in order to provide a flexible system to end users. From there, we need to find the less (energy/time) costly scheduling from both the user and the provider standpoints.

The topic of energy reduction has been intensively studied in the literature and is being investigated at all levels of system abstraction, from the physical layout to software design. There have been several contributions on energy saving focused on scheduling/processors [4, 5, 11, 12, 13], data organizations [14, 6], compilation [22, 23, 24, 30], and the algorithmic level [26, 27, 30, 5]. Power management in sensors network, where energy is really critical,

is addressed in [2]. The research at the architecture level has led to new and advanced low energy architectures, like the Mobile SDRAM and the RDRAM, that support several low power features such as multiple power states of memory banks with dynamic transitions [16, 17], row/column specific activation, partial array refresh, and dynamic voltage/frequency scaling [25]. Current and future generation processors have their clock frequency that can be dynamically modified, and some of them are equipped with a sensor to measure the temperature. In addition, the upper threshold temperature beyond which the fan is automatically triggered can be dynamically adjusted too. But all these need to be soundly monitored. This could be done statically at compile time, or dynamically at runtime.

Although power is proportional to the speed cubed[1], it is known that an important part of energy dissipation comes from memory activities [6, 7], sometimes more that 90% [17]. Consequently, the topic of memory energy reduction is also into the spotlight. For the purpose of reducing the energy dissipation, contributions on cache memory optimization can be considered because of the resulting reduction in memory accesses [8, 9, 10, 27]. In order to benefit from the availability of different memory operating modes, effective memory controller policies should suit the tradeoff between the energy reduction obtained from the use of low power modes and the energy overhead of the consequent activations (*exit latency and synchronization time*) [31]. A combinatorial scheduling technique is proposed by Tadonki et al [29]. A *threshold* approach is considered by Fan et al. [31] in order to detect the appropriate instant for transitions into low power modes. A hardware-assisted approach for detecting and estimating idleness in order to perform power mode transitions is studied by Delaluz et al [17]. We now describe our contribution on the topic of power minimization related to memory accesses and storage.

4.3 An analytical model for energy minimization

4.3.1 Summary

The goal of this work is to design and evaluate a formal model for the energy minimization problem. This is important as a first step toward the design of an efficient power management policy. Our model clearly shows the relative impact of the storage cost and the activation overhead. The optimization problem derived from our model is a quadratic programming problem, that is well solved by standard routines. We consider only the transitions from low power modes to the active mode, so we say *activation* instead of *transition*. Given a predetermined amount of activations to be performed, our model gives the optimal assignment among the different power modes and the corresponding fraction of time that should be spent in each mode. It is clear that there is a correlation between the number of activations and the time we are allowed to spent in each mode. It is important to assume that the time we spend in a low power mode after a transition is bounded. Otherwise, we should transition to the lowest power mode and stay in that mode until the end of the computation. This is unrealistic in general because memory accesses occur very often following an unpredictable pattern, and each memory access triggers a transition to the active mode. To capture this aspect, we consider a time slot for each power mode. Each transition to a given power mode implies that we will spent a period of time that is in a fix range (parameterizable). Once those parameters are determined, the associated energy can be expressed as a function of the total activations pattern. Our goal is then to minimize this objective function and provide the optimal number of transitions for each mode.

4.3.2 A model of energy evaluation

We assume that the energy spent for running an algorithm depends on three major types of operation:

- ◇ the operations performed by the processor (arithmetic and logical operations, comparisons, etc...);
- ◇ the operations performed on the memory (read/write operations, storage, and state transition);
- ◇ the data transfers at all levels of hardware system.

In this paper, we will focus only on the energy consumed by memory operations. We consider the memory energy model defined in [26], which we restate here. The memory energy $E(n)$ for problem size n is defined as the sum of the memory access energy, the data storage energy, and state transition overheads. This yields the formula

$$E(n) = K_a \times C(n) + K_s \times S(n) \times A(n) + K_p \times P(n), \tag{4.1}$$

where

- ◇ K_a is the access energy cost per unit of data, and $C(n)$ represents the total number of memory accesses
- ◇ K_s is the storage energy cost per unit of data per unit time, $S(n)$ is the space complexity, and $A(n)$ is the total time for which the memory is active
- ◇ K_p is the energy overheads for each power transition, and $P(n)$ represents the total number of state transition.

As we can see, the model consider two memory state (*active* and *inactive*), and a single memory bank. Moreover, the storage cost in intermediate modes is neglected, otherwise we should have considered $T(n)$ (the total computation time) instead of $A(n)$ (the total active time). In our paper, we consider the general case with any given number of memory states, and several memory banks with an independent power control.

The main memory $\mathcal{M}$ is composed of p banks, and each bank has q possible inactive states. We denote the whole set of states by $\mathcal{S} = \{0, 1, 2, \cdots, q\}$, where 0 stands for the *active* state. For state transition, we consider only the activations (transition from a low power mode to the active node). This is justified by the fact that transitions to low power modes impact a negligible energy dissipation. The activation energy overheads is given by the vector $W = (w_0, w_1, \cdots, w_q)$, $w_0 = 0$. During the execution of an algorithm, a given bank i spends a fraction α_{ij} of the whole time in state j, thus we have

$$\sum_{j=0}^{q} \alpha_{ij} = 1. \tag{4.2}$$

About the storage cost, let $Q = (q_j), j = 0, \cdots, q$ denotes the vector of storage cost, means q_j is the storage cost per unit data and per unit time when the memory is in power state j.

Concerning the activation complexity, note that since activations occur in a sequential processing, and the transition cost does not depend on the memory bank, we only need to

consider the number of activations from each state j, we denote x_j. We then define the activation vector $x = (x_0, x_1, \cdots, x_q)$.

If we assume that memory banks are of same volume α, we obtain the following memory energy formula for problem size n

$$E(n) = K_a \times C(n) + T(n) \times (\sum_{i=1}^{p}\sum_{j=0}^{q} \alpha_{ij} q_j) \times \alpha + \sum_{j=0}^{q} x_j w_j. \tag{4.3}$$

We define the vector $y = (y_0, y_1, \cdots, y_q)$ by

$$y_j = \sum_{i=1}^{p} \alpha_{ij}. \tag{4.4}$$

For a given state j, y_j is the accumulation of the fractions of time each memory bank has spent in mode j. In case of a single memory bank, it is the fraction of the total execution time spent in the considered mode. The reader can easily see that

$$\sum_{j=1}^{q} y_j = p. \tag{4.5}$$

We shall consider the following straightforward equality

$$\sum_{i=1}^{p}\sum_{j=0}^{q} \alpha_{ij} q_j = \sum_{j=0}^{q} (\sum_{i=1}^{p} \alpha_{ij}) q_j = yQ^T.$$

We define the vector $H = (H_0, H_1, \cdots, H_q)$ as the vector of activation delays, H_j is the time overhead induced by an activation from state j ($H_0 = 0$).
The total time $T(n)$ is composed of

- ◇ the cpu time $\tau(n)$
- ◇ the memory accesses time $\delta C(n)$ (δ is the single memory access delay)
- ◇ the activations overhead Hx^T

We can write

$$E(n) = K_a \times C + \alpha \times (\tau + \delta C + Hx^T) \times yQ^T + xW^T. \tag{4.6}$$

We make the following considerations

- ◇ the power management energy overhead xW^T is negligible [26].
- ◇ the the additive part $K_a \times C(n)$ can be dropped since it doesn't depend on the power state management.

Thus, the objective to be minimized is (proportional to) the following

$$E(x, y) = [Hx^T + (\tau + \delta C)] yQ^T. \tag{4.7}$$

4.3.3 Optimization

Problem Formulation

Our goal is to study the energy reduction through the minimization of the objective (4.7). In order to be consistent and also avoid useless (or trivial) solutions, a number of constraints should be considered

Domain specification. The variables x and y belong to $\mathcal{N}$ and $\mathcal{R}$ respectively, i.e.

$$x \in \mathbb{N}^q, \tag{4.8}$$
$$y \in \mathbb{R}^q. \tag{4.9}$$

Time consistency. As previously explained, we have

$$y \geq 0, \tag{4.10}$$
$$y_1 + y_2 + \cdots + y_q = p, \tag{4.11}$$

Another constraint that should be considered here is related to the fraction of time spent in the active mode (y_0). Indeed, the time spent in the active mode is greater than the total memory access time, which can be estimated from the number of memory accesses C, and the time of a single access δ. Since, we consider fraction of time, we have

$$y_0 \geq \frac{\delta C}{R}, \tag{4.12}$$

where δC is the total memory access time, and R the total running time (without the power management overhead) which can be estimated from the time complexity of the program or from a profiling.

Activations bounds. It is reasonable to assume that each time a memory bank is activated, it will earlier or later be accessed. Thus, we have

$$\sum_{i=0}^{q} x_i \leq C. \tag{4.13}$$

However, except the ideal case of a highly regular and predictable memory access, several activations should be performed for a better use of power modes availability. This is well captured by a lower bound the number of activations. Thus, we have a lower bound and an upper bound in the number of activation. In our model we consider a fix amount of activations instead of a range. This gives,

$$x_1 + x_2 + \cdots + x_q = \rho C, \tag{4.14}$$

where ρ is a scaling factor such that $0 \leq \rho \leq 1$.

Compatibility between time and activation. Recall that a memory bank is activated if and only if it will be accessed. Moreover, when a memory bank is put in a given low power mode, a minimum (resp. maximum) period of time is spent in that mode before transitioning to the active mode. This can be the fraction of time taken by the smallest job (or instruction

depending on the granularity). We consider the set of time intervals $[\varphi_i, \eta_i]$ low power modes. Then, we have

$$\varphi_j x_j \leq y_j \leq \eta_j x_j \text{ for } j = 1, \cdots, q. \tag{4.15}$$

In addition, since any of every activation implies a minimum period of time, we denote γ, in the active mode, we also have

$$y_0 \geq \gamma(\sum_{j=0}^{q} x_j). \tag{4.16}$$

Using relation (4.14), relation (4.16) becomes

$$y_0 \geq \gamma\rho C. \tag{4.17}$$

We shall consider μ define by

$$\mu = \max\{\frac{\delta}{R}, \gamma\rho\}. \tag{4.18}$$

The inequalities (4.12) and (4.17) can be combined to

$$y_0 \geq \mu C. \tag{4.19}$$

We now analyze the model.

4.3.4 Model analysis

We first note that transitioning from the active state to state j for a period of time Δt is advantageous (based of storage cost) if and only if we have

$$q_j(\Delta t + h_j) \leq q_0 \Delta t, \tag{4.20}$$

which gives the following threshold relation

$$\Delta t \geq (\frac{q_j}{q_0 - q_j})h_j. \tag{4.21}$$

The time threshold vector D defined by

$$D_j = (\frac{q_j}{q_0 - q_j})h_j, j = 1, 2, \cdots, q \tag{4.22}$$

provides the minimum period of time that should be spent in each low power modes, and is also a good indicator to appreciate their relative impact. We propose to select the time intervals (4.15) for low power modes as follows

$$\varphi_j = \lambda_1 \frac{D_j}{R} \quad \varphi_j = \lambda_2 \frac{D_j}{R}, \tag{4.23}$$

where $1 \leq \lambda_1 \leq \lambda_2$.
Lastly, the active time threshold as defined in (4.17) should be greater than the memory accesses time. Then we should have

$$\gamma \geq \frac{\delta}{\rho R}. \tag{4.24}$$

We now solve the optimization problem provides by our model as described above.

Solving the Optimization Problem

According to our model, the optimization problem behind the energy reduction is the following

$$\texttt{min}\ xH^TQy^T + RQy^T$$
$$\texttt{subject to}$$
1. $x \in \mathbb{N}^q$,
2. $y \in \mathbb{R}^q$,
3. $y_1 + y_2 + \cdots + y_q = p$,
4. $y_0 \geq \mu C$,
5. $x_1 + x_2 + \cdots + x_q = \rho C$,
6. $y \leq \varphi x$.
7. $y \geq \eta x$.

Figure 4.1: Energy minimization problem

There are mainly two ways for solving the optimization problem formulated in figure 4.3.4. The first approach is to consider the problem as a mixed integer programming problem (MIP). For a given value of x, the resulting model becomes a linear programming (LP) problem. Thus, appropriate techniques like the standard LP based Branch and Bound can be considered. However, we think that this is an unnecessarily challenging computation. Indeed, a single transition does not have a significant impact on the overall energy dissipation as quantified by our model. Thus, we may consider a pragmatic approach where the variable x is first assumed to be continuous, and next rounded down in order to obtain the required solution. This second approach yields a simple quadratic programming model that is easily solved by standard routines.

4.3.5 Experiments

We evaluate our model with the values provided in [31] for the RDRAM. Table 4.1 summarizes the corresponding values (vector D is calculated using the formula (4.22)). Our optimization is performed with MATLAB through the code listed below.

```
Q = (300 180 30 3)
H = (0 16 60 6000)
p = 8
q = 4
δ = 60
D = (9.00 6.67 60.61)
```

Table 4.1: DRAM settings

```
function [X,Y,E] = Energy_Opt(H,Q,R,d,C,p,q,r,g,l1,l2)
% Matlab code to solve the energy minimization problem
% The quadratic objective is considered as follows
% 0.5 * X' * HH * X + ff' * X

% We form our objective coefficients
HH = [zeros(q, q), H' * Q; Q' * H , zeros(q, q)];
ff = R * [zeros(q, 1); Q'];

% Bound on the main variable Z = [X,Y]
LB = [zeros(q, 1) ; zeros(q, 1)];
UB = [inf * ones(q, 1) ; p * ones(q, 1)];
% Ajust the lower bound on Y1 (Y0 in the text)
LB(q+1) = max(d * C / t, g * r * C);

% Matrix of the equality constraints
Aeq = [ones(1,q), zeros(1,q); zeros(1,q) , ones(1,q)];
beq = [r * C; p];

% Matrix of the inequality constraints
a1 = [l1 * diag(D), - eye(q)]; b1 = zeros(q, 1);
a2 = [-l2 * diag(D),  eye(q)]; b2 = zeros(q, 1);
% Y0 is not bounded by X
a1(1,:)=[]; b1(1)  =[]; a2(1,:)=[]; b2(1)  =[];
% Forming the matrix
A  = [a1; a2];  b  = [b1; b2];

% OPTIMIZATION unsing the solver quadprog of MATLAB
[Z, E, EXITF,OUTPUT]  = quadprog(HH,ff,A,b,Aeq,beq,LB,UB);

% RETRIVING X AND Y from Z
X  = Z(1:q);
Y  = Z(q + 1: 2 * q);
```

We consider a problem (abstracted) where 75% of the total time is spent in memory accesses. We used `R = 80000` and `C = 1000`. Note that our objective function is proportional to the time vector y and the vector of storage coefficient Q. Thus, the measuring unit can be scaled as desired without changing the optimal argument. Table 4.2 displays a selection of optimal activation repartition and the percentage of energy that is saved or lost. Figure 4.2 shows how the energy varies in relation with the number of activations.

ρ	N_{act}	X	Y	E_{opt}	Reduction
0	0	(0, 0, 0, 0)	(8, 0, 0, 0)	1.92	0%
0.01	10	(0, 10, 0, 0)	(7.58, 0, 0.42, 0)	1.84	4%
0.02	20	(0, 0, 7, 13)	(3, 0, 0.31, 4.69)	1.43	25%
0.05	50	(0, 0, 41, 9)	(3, 0, 1.71, 3.29)	1.30	32%
0.10	100	(0, 0, 97, 3)	(3, 0, 4.04, 0.96)	1.04	46%
0.11	110	(0, 8, 100, 2)	(3, 0.1, 4.15, 0.74)	1.024	47%
0.125	125	(0, 25, 100,0)	(3, 0.85, 4.15, 0)	1.014	48%
0.20	200	(0, 100, 100, 0)	(3, 1.3, 3.7, 0))	1.08	44%
0.21	210	(0, 100, 100, 10)	(3, 1.3, 0.83, 2.87)	1.72	-11%
0.22	220	(0, 100, 100, 20)	(3, 1.3, 0.83, 2.87)	2.41	-26%
0.25	250	(0, 100, 100, 25)	(3, 1.3, 0.83, 2.87)	2.76	-44%

Table 4.2: Experiments with our model on a RDRAM

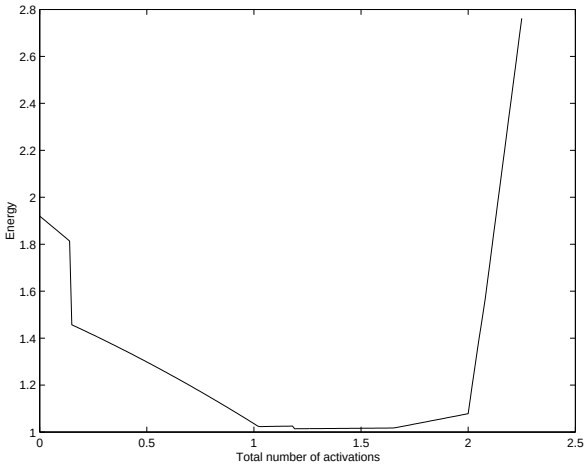


Figure 4.2: Energy vs the number of activations

As we can see from Table 4.2, the best number of activation is 125 (12.5% of the number of memory accesses), with an energy reduction of 48% (taken the always active case as baseline). We also see that there is a critical value for the number of activations (200 in this case) under which we begin loosing energy. In addition, the optimal distribution of activations among low power modes depends on the total number of activations and the time we are allowed to stay in each mode.

4.3.6 Conclusion

We have formulated the problem of energy optimization in the context of several low power modes. We have shown that, in order to make a rewarding transition to a given low power mode, there is a minimum period of time that should be spent in that mode. From our experiments with a RDRAM, it follows that a reduction of 48% can be obtained by performing regular transitions. The optimal number of activations is determined experimentally. We think that our model can be used for a first evaluation of potential energy reduction before moving forward to any power management policy.

Bibliography

[1] D. M. Brooks, P. Bose, Stanley E. Schuster, H. Jacobson, P. N. Kudva, A. Buyuktosunoglu, J.-D. Wellman, V. Zyuban, M. Gupta, and P. W. Cook, *Power aware microarchitecture: Design and modeling challenges for next-generation microprocessors*, IEEE Micro, 20(6):26.44, 2000.

[2] C. Tadonki and J. Rolim, *An integer programming heuristic for the dual power management problem in wireless sensor networks*, 2nd International Workshop on Managing Ubiquitous Communications and Services, MUCS2004, Dublin, Ireland, December 13, 2004.

[3] S. Albers, F. Mller, and S. Schmelzer, *Speed scaling on parallel processors*, 19th ACM Symposium on Parallelism in Algorithms and Architectures (SPAA'07), San Diego, CA, USA , 2007.

[4] S. Albers, H. Fujiwara, *Energy-Efficient Algorithms for Flow Time Minimization*, Symposium on Theoretical Aspects of Computer Science (STACS), Marseille, France, 2006.

[5] S. Irani and K. R. Pruhs, *Algorithmic problems in power management*, SIGACT News, 36(2):6376, 2005.

[6] F. Catthoor, S.Wuytack, E.D. Greef, F. Balasa, L. Nachtergaele, and A. Vandecappelle, *Custom memory management methodology - exploration of memory organization for embedded multimedia system design*, Kluwer Academic Pub., June 1998.

[7] A. R. Lebeck, X. Fan, H. Zeng, and C. S. Ellis, *Power aware page allocation*, Int. Conf. Arch. Support Prog. Lang. Ope. Syst., November 2000.

[8] M. B. Kamble and K. Ghose, *Analytical energy dissipation models for low power caches*, Int. Symp. Low Power Electronics and Design, 1997.

[9] W-T. Shiue and C. Chakrabarti, *Memory exploration for low power embedded systems*, Proc. DAC'99, New Orleans, Louisina, 1999.

[10] C. Su and A. Despain, *Cache design trade-offs for power and performance optimization: a case study*, In Proc. Int. Symp. on Low Power Design, pp. 63-68, 1995.

[11] D. Brooks and M. Martonosi, *Dynamically exploiting narrow width operands to improve processor power and performance*, In Proc. Fifth Intl. Symp. High-Perf. Computer Architecture, Orlando, January 1999.

[12] V. Tiwari, S. Malik, A. Wolfe, and T. C. Lee, *Instruction Level Power Analysis and Optimization of Software*, Journal of VLSI Signal Processing Systems, Vol 13, No 2, August 1996.

[13] M. C. Toburen, T. M. Conte, and M. Reilly, *Instruction scheduling for low power dissipation in high performance processors*, In Proc. the Power Driven Micro-Architecture Workshop in conjunction with ISCA'98, Barcelona, June 1998.

[14] W. Ye, N. Vijaykrishnan, M. Kandemir, and M. J. Irwin, *The design and use of SimplePower: a cycle-accurate energy estimation tool*, In Proc. Design. Automation Conference (DAC), Los Angeles, June 5-9, 2000.

[15] Todd Austin, *Simplescalar*, Master's thesis, University of Wisconsin, 1998.

[16] 128/144-MBit Direct RDRAM Data Sheet, Rambus Inc., May 1999.

[17] V. Delaluz and M. Kandemir and N. Vijaykrishnan and A. Sivasubramaniam and M. Irwin. Memory energy management using software and hardware directed power mode control. Tech. Report CSE-00-004, The Pennsylvania State University, April 2000.

[18] W. Wolf. Software-Hardware Codesign of Embedded Systems. In , *Proceedings of the IEEE* , volume 82 , 1998.

[19] R. Ernst. Codesign of Embedded Systems: Status and Trends . In , *IEEE Design and Test of Computers* , volume 15 , 1998.

[20] Manfred Schlett. Trends in Embedded Microprocessors Design. In , *IEEE Computer*, 1998.

[21] "Mobile SDRAM Power Saving Features," Technical Note TN-48-10, MICRON, http://www.micron.com

[22] W. Tang, A. V. Veidenbaum, and R. Gupta. Architectural Adaptation for Power and Performance. In , *International Conference on ASIC*, 2001 .

[23] L. Bebini and G. De Micheli. Sytem-Level Optimization: Techniques and Tools. In , *ACM Transaction on Design Automation of Electronic Systems*, 2000.

[24] T. Okuma, T. Ishihara, H. Yasuura . Software Energy Reduction Techniques for Variable-Voltage Processors. In , *IEEE Design and Test of Computers* , 2001 .

[25] J. Pouwelse, K. Langendoen, and H. Sips, "Dynamic Voltage Scaling on a Low-Power Microprocessor," UbiCom-Tech. Report, 2000.

[26] M. Singh and V. K. Prasanna . Algorithmic Techniques for Memory Energy Reduction. In , *Worshop on Experimental Algorithms*, Ascona, Switzerland, May 26-28, 2003.

[27] S. Sen and S. Chatterjee . Towards a Theory of Cache-Efficient Algorithms . In *SODA*, 2000 .

[28] C. Tadonki and J. Rolim , *An analytical model for energy minimization*, III Workshop on Efficient and Experimental Algorithms, WEA04 (LNCS/Springer), Angra dos Reis, Rio de Janeiro, Brazil, May 2004.

[29] C. Tadonki, J. Rolim, M. Singh, and V. Prasanna. *Combinatorial Techniques for Memory Power State Scheduling in Energy Constrained Systems*, Workshop on Approximation and Online Algorithms (WAOA), WAOA2003, Budapest, Hungary, September 2003 .

[30] D.F. Bacon, S.L. Graham, and O.J. sharp . Compiler Transformations for High-Performance Computing . *Hermes*, 1994 .

[31] X. Fan, C. S. Ellis, and A. R. Lebeck. Memory Controller Policies for DRAM Power Management. *ISLPED'01*, August 6-7, Huntington Beach, California, 2001.

Chapter 5

Hybrid supercomputing

5.1 Abstract

With the advent and the pervasiveness of the multi-core technology, which stands now as a standard for processor design, the supercomputing landscape is currently dominated by multi-level heterogeneous supercomputers. Many well-established computing vendors use to announce, design, and promote chips with an increasing number of processor cores. However, most of existing programs are still written from the purely distributed memory basis. Designing programs that mix up both the distributed memory model and the shared memory model is the way go, especially on large hybrid supercomputers.This requires and algorithmic efforts to express and quantify all levels of parallelism, and then find the best way to schedule the tasks on the computing nodes accordingly. On a single multi-core node, there is a complex hierarchical memory system, which should be carefully taken into account in order to avoid a severe performance penalty. All these have brought a noticeable level of complexity in hybrid program design, especially if scalability and good absolute performance are expected. We discuss this topic in the current chapter and provide one case study of our contribution [10].

5.2 Overview of hybrid supercomputing

When it comes to supercomputer, the main problem is *scalability*. The complexity of the communication pattern increases with the number of processors, thus exacerbating the gap between the virtual topology and the physical interconnect. Supercomputers are generally made with shared memory computing nodes with several cores. Ordinary programmers use to consider the processor core as the basic processing unit, and then launch a pure message passing program onto the machine. Current implementation of MPI allows this to work seamlessly, but a scalability wall is quickly reached. Having a shared memory implementation on each multi-core node has several advantages. The first one is that the overall memory of the computing node is available for the task assigned to the node, this also reduces data dependencies. Secondly, the cores within a node do no longer need to exchange data through the network, they concurrently access their local shared memory instead. Third, the global communication topology becomes lighter, this might lead to a significant reduction of the communication cost.

5.3 Special focus on memory hierarchy

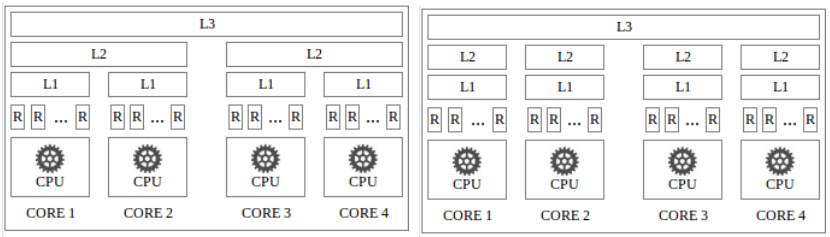
5.3.1 Overview

Multi-core is the standard architecture model for actual and next generation processors [1, 17, 7, 8]. As the main trend is to increase the number of cores on a single chip, the associated hierarchical memory system is becoming more complex and less predictable. Number of critical scientific computation kernels suffer from a heavy memory load, which acts as a bottleneck and likely bounds the overall performance. The case of stencil computation is notoriously severe, because most of local dependences yield neighbors which are distant enough to break the data locality comfort. A plethora of contributions can be found in the literature about ways to improve data locality, with the aim of reaching an optimal cache memory benefit. As long as the primary cache is the main concern, probably in a sequential computation, the problem can be tackled by means of purely topological considerations. However, when it comes to multi-level caches in a multi-core computation, different other aspects need to be

taken into account because of the elusive data movement between caches and the concurrency in the memory accesses.

5.3.2 Multi-level Memory Model

A multi-level memory in a multi-core architecture is typically organized as shown in Figure 5.1.



(a) Shared L2 cache (b) Private L2 cache

Figure 5.1: Canonical multi-level memory

The overall memory system is organized into several hierarchical levels (typically 2 or 3) ranging from the main memory to the primary cache, with a decreasing (resp. increasing) size (resp. access speed). This multi-level organization is mainly justified by the correlation between the size of a memory device and its access latency, as it is for the use of a cache itself. Each core has its own primary cache, namely L1. The second level of cache, namely L2, is either shared by pair of cores (Figure 5.1 (a)) or is private too (Figure 5.1 (b)). The third level of cache (whenever exists) is shared by all four cores. Typically, the basic building block is a CPU die with $2p$ cores, where L1 is always private, L2 is private (Figure 5.1(a)) or attached to a pair of cores (Figure 5.1(b)) , and L3 is shared by all cores. For case (a), two examples are the 6-cores Intel Dunnington and he 4-cores IBM Power5. For case (b), now more common, two examples Intel Xeon and AMD Opteron series A multiprocessor is made of a replication of several CPUs, connected via special bus controllers like the QuickPath Interconnect (QPI). The global (main) memory is of course shared by all processor cores and is generally NUMA. Figure 5.2 illustrates the standard CPUs packaging.

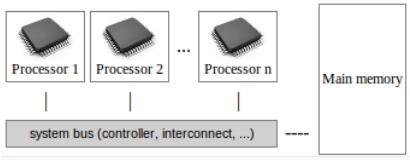


Figure 5.2: CPUs packaging

For sake of simplicity, it is common to consider a model where both the L3-cache and the main memory can be accessed by any core in a uniform basis (which is not true!). Because upper memory levels experience wider sharing, each access might incur an additional delay due to *arbitration*, *snooping for requests* or *coherency*, and *bus contention*, to name a few. The so-called *false sharing* is just a (somehow frustrating) consequence of a strict coherency protocol. When it comes to benefit from the cache, sharing also has also number of drawbacks, which mainly come from unbalanced or disorderly concurrent accesses.

Shared multi-level memories are technically complex. Current compilers do not address the concurrency and the multi-level aspects of the memory, nor the multi-threading of the sequential code. Thus, it is up to the programmer to design his code accordingly. We now describe a case study of a hybrid implementation on a supercomputer.

5.4 Large scale Kronecker product on supercomputers

5.4.1 Abstract

The *Kronecker product*, also called *tensor product*, is a fundamental matrix algebra operation, which is widely used as a natural formalism to express a convolution of many interactions or representations. Given a set of matrices, we need to multiply their Kronecker product by a vector. This operation is a critical kernel for iterative algorithms, thus needs to be computed efficiently. In a previous work, we have proposed a cost optimal parallel algorithm for the problem, both in terms of floating point computation time and interprocessor communication steps. However, the lower bound of data transfers can only be achieved if we really consider (local) logarithmic broadcasts. In practice, we consider a communication loop instead. Thus, it becomes important to care about the real cost of each broadcast. As this local broadcast is performed simultaneously by each processor, the situation is getting worse on a large number of processors (supercomputers). We address the problem in this work in two points. On one hand, we propose a way to build a virtual topology which has the lowest gap to the theoretical lower bound. On the other hand, we consider a hybrid implementation, which has the advantage of reducing the number of communicating nodes. We illustrate our work with some benchmarks on a large SMP 8-Core supercomputer.

5.4.2 Introduction

The *Kronecker product* is a basic matrix algebra operation, which is mainly used for multidimensional modeling in number of specialized fields[5]: *Stochastic Automata Networks* (SAN) [3, 4, 5], *Fast Fourier Transform* (FFT), *Fast Poisson Solver* (FPS) [13, 14], *Quantum Computation* (QC) [9] and *Lattice Quantum Chromodynamics* [12]. Considering a principal matrix expressed as a Kronecker product of several matrices, iterative schemes require to repeatedly multiply such a matrix by a vector. Formally, we are given N square matrices $A^{(i)}$ of sizes n_i, $i = 1, \cdots, N$, and a vector x of length $L = n_1 n_2 \cdots n_N$, and we need to compute y (of length L) given by

$$y = x(\bigotimes_{i=1}^{N} A^{(i)}). \tag{5.1}$$

It is well-known that we should not compute the matrix explicitly before performing the multiplication, as this would require a huge memory to store that matrix and will yield redundant computations. A cost optimal algorithm for this computation proceeds in a recursive way, consuming one matrix $A^{(i)}$ after another [11]. Consequently, traditional parallel routines for matrix-vector product cannot be considered. When starting with the recursive algorithm as a basis, any parallel scheme will involve a set of data communication at the end of each iteration. The cost of this communication is the main challenge for this problem, especially with a large number of processors, because there is a significant interleave between the (probably

virtual) communication links. Moreover, in order to reduce the cache misses due to an increasing stride from one iteration to the next one, array reshuffling is sometimes considered, and this complicates the communication topology.

In [11], we have proposed an efficient parallel algorithm which achieves the multiplication without explicit shuffling and requires a minimal number of communication steps. However, the real cost of each communication step depends on the virtual topology and the way the transfers are really performed. This problem was left open in this work because of the modest size of the parallel computers considered (up to 256 processors). In this report, we provide an algorithm to construct an efficient topology, in addition to a hybrid implementation using OpenMP[15] on the computing multicore nodes. With this contribution, we keep the global efficiency of the original algorithm on a larger number of processors as illustrated by some benchmark results. The rest of the report is organized as follows. Section 5.4.3 gives an overview of the original algorithm. This is followed in section 5.4.4 by a discussion on its complexity and the position of the problem. We describe our heuristic to find an efficient topology in section 5.4.5. We discuss the hybrid implementation and section 5.4.6. In section 5.4.7, we display and comment our benchmark results. We conclude in section 5.4.8.

5.4.3 Original parallel algorithm

We restate our parallel algorithm in order to provide a self-contained material, the reader could refer to [11] for more details. From (5.1) and using the so-called *canonical factorization*, we obtain the recursive scheme defined by (5.2)

$$\begin{cases} V^{(N+1)} = x \\ V^{(s)} = V^{(s+1)}(I_{n_1 \cdots n_{s-1}} \otimes A^{(s)} \otimes I_{n_{s+1} \cdots n_N}) \end{cases} \tag{5.2}$$

which leads at the last step to $V^{(1)} = x \otimes_{i=1}^{N} A^{(i)}$. Our parallelization of the recursive computation expressed by equation (5.2) can be defined as follows. Given p processors (assuming that p divides $L = n_1 n_2 ... n_N$), we proceed as follows. We first compute a sequence of N integers p_i such that $p = p_1 p_2 ... p_N$ and p_i divides $n_i, i = 1, 2, ..., N$. Considering a multidimensional indexation, we say that each processor $(a_1, a_2, \cdots, a_N)$ computes the entries $(b_1, b_2, \cdots, b_N)$ of $V^{(s)}$ such that $b_i \bmod p_i = a_i,\ i = 1, 2, \cdots, N$. A complete description of the parallel algorithm is given by Alg. 1. Note that the *send* and *receive* occurrences can be combined into a single *sendreceive* call because of the symmetry of the topology.

5.4.4 Communication complexity

Our scheduling onto p processors is based on a decomposition $(p_1, p_2, \cdots, p_N)$ such that p_i divides n_i, and $p_1 p_2 \cdots p_N = p$. In theory, algorithm Alg. 1 performs $log(p)$ parallel communication steps when executed with p processors. Indeed, one local broadcast occurs at the end of each step s, thus we do $log(p_1) + log(p_2) + \cdots + log(p_N) = log(p_1 p_2 \cdots p_N) = log(p)$ parallel communication steps. This assumes that, at a given step i, we perform $log(p_i)$ parallel transfers (local broadcast to p_i processors by each processor). However, in practice, we issue $p_i - 1$ transfers (communication loop). Thus, the gap between $p_i - 1$ and $log(p_i)$ becomes important for larger p_i. Actually, each processor performs $p_1 + p_2 + \cdots + p_N$ transfers in total. On a larger cluster, there will be an additional overhead coming from the gap between the virtual topology and the physical topology. We first focus on how to find a decomposition which reduces the measure $p_1 + p_2 + \cdots + p_N$.

```
π ← 1; r ← 1; ℓ ← c_1c_2...c_N = L/p     /*
c_i = n_i/d_i */
y ← x(Q_{1w_1}, Q_{2w_2}, ..., Q_{Nw_N})
For s ← N downto 1 do
  ℓ ← ℓ/c[s]
  ws = [w div(π)] mod(d[s]) + 1
  e ← (ws − 1) × c[s]
  v ← 0
  i ← 1
  For a ← 1 to ℓ do
    For j ← e + 1 to e + c[s] do
      For b ← 1 to r do
        For t ← e + 1 to e + c[s] do
          v[i] ← v[i] + A(s, t, j)y[I + (t − j)r]
        end do
        i ← i + 1
      end do
    end do
  end do
  If (ws = 1) then H ← d else H ← ws − 1
  For T = ws + 1 to ws + d[s] − 1 do
    G ← mod(T − 1, d[s]) + 1
    idest ← w + (G − ws) × π
    isender ← w + (H − ws) × π
    send(y, idest, ws)
    recv(u, isender, H)
    e ← (H − 1) × c[s]
    i ← 1
    For a ← 1 to ℓ do
      For j ← 1 to c[s] do
        For b ← 1 to r do
          For t ← e + 1 to e + c[s] do
            v[i] ← v[i] + A(s, t, j)u[I + (t − j)r]
          end do
          i ← i + 1
        end do
      end do
    end do
   If (H = 1) then H ← d[s] else H ← H−1
  end do
  r ← r × c[s]
  π ← π × d[s]
  If (s > 1) then y ← v
end do
z(Q_{1w_1}, Q_{2w_2}, ..., Q_{Nw_N}) ← v
```

Alg. 1 : Implementation of the matrix-vector product.

5.4.5 Heuristic for an efficient topology

We propose the algorithm Alg. 2 to find an efficient decomposition for a given number of processors p, which is a factor of $n_1 n_2 \cdots n_N$.

$d \leftarrow p$
{Starting decomposition }
For $i \leftarrow 1$ **to** N **do**
  $p_i \leftarrow \gcd(d, n_i)$
  $d \leftarrow \frac{d}{p_i}$
enddo
{Recursive refinement }
For $i \leftarrow 1$ **to** N **do**
  For $j \leftarrow 1$ **to** N **do**
    $\alpha \leftarrow \gcd(p_i, \frac{n_j}{p_j})$
    if $((\alpha > 1) \wedge (p_i > cp_j))$
      $p_i \leftarrow \frac{d_i}{\alpha}$
      $p_j \leftarrow \alpha d_j$
    endif
  enddo
enddo

Alg. 2 : Heuristic for an efficient decomposition

The principle of Alg. 2 is the following. We start with a *gcd* decomposition. Next, we refine it using the fact that if $p_i > \alpha p_j$, with α a non trivial factor of p_i, then $p_i/\alpha + \alpha p_j < p_i + p_j$. It is thus rewarding to replace p_i (resp. p_j) by p_i/α (resp. αp_j). Once this is done, it is clear that on a larger cluster (i.e. large value of p), all these simultaneous transfers will exacerbate the communication overhead and certainly slowdown the global performance. Fortunately, most modern supercomputers are built up with multicore nodes. Thus, a hybrid implementation, which combines the standard distributed memory implementation with a shared memory program (SMP) on the nodes, will overcome the problem by reducing the number of communicating nodes.

5.4.6 SMP implementation

We chose to use OpenMP to derive our shared memory code. Looking at Alg. 1, we decide to put the loop distribution pragma over the a loop. In order to do so, we first need to remove the $i \leftarrow i + 1$ incrementation and directly calculate the i index, which is given by $i \leftarrow c[s] \times r \times (a-1) + r \times (j-1) + b$. Now, the length ℓ where the loop blocking will occur varies with s ($\ell \leftarrow \ell / c[s]$). Thus, we need to keep it being a factor of the (fixed) number of threads. We achieve it by splitting the main loop into two parts, means isolating the case $s = N$ and then enclose the rest ($s = N-1, N-2, \cdots, 1$) into a parallel section. Moreover, since the number of nodes is now reduced to p/T (T is the number of OpenMP threads), we need to adapt our primarily decomposition such that ℓ remains a factor of T. The general way to do that is to split the loop over s at the right place (not only the extremal iteration), but this would be better implemented with Posix threads library, because we could dynamically manage the threads to handle desired loop partitioning (this is left for future work). We now show the impact of our strategy on benchmark results. Interested reader can download the

source code at
http://www.omegacomputer.com/staff/tadonki/codes/kronecker.f

5.4.7 Experimental results

We consider a SMP 8-core cluster named JADE [16]. The whole cluster JADE is composed of 1536 compute nodes (i.e. 1536 x 8 = 12288 cores of Harpertown type processors) and 1344 compute nodes of nehalem type processor (1344 x 8 = 10 752 cores). The network fabric is an Infiniband (IB 4x DDR) double planes network for the first part of the machine (Harpertown), whereas 4 drivers InfiniBand 4X QDR provide 72 ports IB 4X QDR on output of each IRU of the second part of the machine (576 Go/s).

We choose $N = 6$ square matrices of orders 20, 36, 32, 18, 24, and 16, which means a principal matrix of order $L = 159\ 252\ 480$. We first show in table 1 the results of the pure MPI code. The decomposition obtained with our algorithm is marked with a star and is surrounded by two alternative decompositions (the one obtained by a basic gcd decomposition and the less distributed one) to illustrate the difference.

p	decomposition	time(s)
32	(4,1,8,1,1,1)	2.06 s
32	(2,2,2,2,2,1)*	1.62 s
32	(1,1,32,1,1,1)	4.14 s
180	(20,9,1,1,1,1)	0.75 s
180	(5,3,2,2,3,1)*	0.34 s
180	(10,6,1,1,3,1)	0.49 s
720	(20,36,1,1,1,1)	1.20 s
720	(10,3,2,2,3,2)*	0.23 s
720	(10,9,4,2,1,1)	0.35 s
2880	(20,36,4,1,1,1)	1.47 s
2880	(10,6,2,2,6,2)*	1.20 s
2880	(20,12,2,2,3,1)	1.32 s
4320	(20,36,2,3,1,1)	1.48 s
4320	(10,3,4,3,3,4)*	0.92 s
4320	(20,18,4,3,1,1)	1.34 s

Table 5.1: MPI implementation timings

From Table 5.1, we see that for a given number of processors, the partition obtained with our procedure can improve the global performance by a factor from 2 to 5 (see $p = 720$). However, when the number or MPI processes increases, we see that we lose the scalability, because data communication severely dominates (the code is cost optimal for floating point operations). We now see how this is improved using a hybrid implementation. We reconsider the previous best decompositions as baseline and compare each of them with the corresponding hybrid configuration. For each number of cores in $\{4320, 2880, 720\}$, we consider a k-cores SMP clustering, $k \in \{1, 4, 8\}$.

#MPI	decomposition	#threads	time	speedup
4320	(10,3,4,3,3,4)	1	0.92 s	1
1080	(5, 3, 2, 3, 3, 4)	4	0.16 s	5.75
540	(5, 3, 2, 3, 3, 2)	8	0.12 s	7.67
2880	(10,6,2,2,6,2)	1	1.20 s	1
360	(5, 3, 2, 3, 3, 4)	8	0.16 s	7.5
720	(10,3,2,3,3,2)	1	0.23 s	1
90	(5, 3, 2, 1, 3, 1)	4	0.45 s	0.51

Table 5.2: Hybrid (MPI+OpenMP) code timings

We can see from Table 5.2 that we are close to a linear (threads) speedup with 4320 and 2880 cores. This is due to the fact that the global computation time was really dominated by data communication and synchronization mechanism. For a smaller number of cores, we see that we start loosing the benefit of the SMP implementation. This is due the (predictable) cache misses penalty coming from the stride $(t-j) \times r$ in Alg. 1, which is increasingly bigger since r does. We could use larger number of cores, but we our experimental configuration sufficiently illustrative of what we need to show and how our solution contributes to the issues.

5.4.8 Conclusion

The problem of multiplying a vector by a Kronecker product of matrices is crucial in stochastic sciences and is a computationally challenging task for large instances. In order to avoid a memory bottleneck and redundant computation, a recursive scheme has been mathematically formulated, for which corresponding efficient implementations are expected. In one hand, the increasing loop stride needs to be handled carefully in order to reduce the impact of caches misses. This aspect really dominates and thus needs to be seriously taken into account in the performance analysis. In the other hand, the parallelization requires an important number of parallel transfers, which could become problematic on large clusters. This paper provides a contribution on both aspects, based on a cost optimal solution (floating point computation point of view) from the literature. Our solution is a combination of a heuristic procedure to build an efficient virtual topology and the use of hybrid programming paradigm. Our experimental results illustrate the improvement of our contribution, and evidence the need of a compromise on large clusters.

Bibliography

[1] K. Asanovic, R. Bodik, B. C. Catanzaro, J. J. Gebis, P. Husbands, K. Keutzer, D. A. Patterson, W. L. Plishker, J. Shalf, S. W. Williams, and K. A. Yelick, *The Landscape of Parallel Computing Research: A View from Berkeley*, Technical Report No. UCB/EECS-2006-183, EECS Department, University of California, Berkeley, 2006.

[2] M. Davio, *Kronecker Products and Shuffle Algebra*, IEEE Trans. Comput., Vol. C-30, No. 2, pp. 116-125, 1981.

[3] P. Fernandes, B. Plateau, and W. J. Stewart, *Efficient Descriptor-Vector Multiplications in Stochastic Automata Networks*, INRIA internal report No. 2935 , July 1996.

[4] A. Benoit, P. Fernandes, B. Plateau, W. J. Stewart, *On the benefits of using functional transitions and Kronecker algebra*, Performance Evaluation, Vol. 58(4), pp. 367-390, 2000.

[5] Benoit, Anne and Plateau, Brigitte and Stewart, William J., *Memory-efficient Kronecker algorithms with applications to the modeling of parallel systems*, Future Gener. Comput. Syst., Vol. 22(7), pp. 838-847, 2006.

[6] J. Granta, M. Conner, and R. Tolimieri, Recursive fast algorithms and the role of the tensor product, IEEE Transaction on Signal Processing, 40(12):2921-2930, December 1992.

[7] P. Kongetira, K. Aingaran, and K. Olokotun. *Niagara: A 32-Way Multithreaded Sparc Processor*, IEEE Micro, 25(2):21-29, March/April 2006.

[8] R. Kota and R. Oehler, *Horus: Large-Scale Symmetric Multiprocessing for Opteron Systems*, IEEE Micro, 25(2):30-40, March/April 2006.

[9] P. W. Shor, Quantum Computing, Proceeding of the ICM Conference, 1998.

[10] C. Tadonki, *Large Scale Kronecker Product on Supercomputers*, 2nd Workshop on Architecture and Multi-Core Applications (WAMCA 2011) in conjunction with the International Symposium on Computer Architecture and High Performance Computing (SBAC PAD 2011), Vitria, Esprito Santo, Brazil, October 26-29, 2011.

[11] C. Tadonki and B. Philippe, Parallel Multiplication of a Vector by a Kronecker Product of Matrices, Journal of Parallel and Distributed Computing and Practices, Parallel Distributed Computing Practices PDCP, volume 3(3), 2000.

[12] C. Tadonki, G. Grosdidier, and O. Pene, *An efficient CELL library for lattice quantum chromodynamics*, ACM SIGARCH Computer Architecture News, Vol. 38(4), 2011.

[13] C.Tong and P. N. Swarztrauber, Ordered Fast Fourier Transforms on Massively Parallel Hypercube Multiprocessor, Journal of Parallel and Distributed Computing 12, 50-59, 1991.

[14] C. Van Loan, *Computational Framework for the Fast Fourier Transform*, SIAM, 1992.

[15] http://openmp.org/

[16] http://www.cines.fr

[17] *From a Few Cores to Many: A Tera-scale Computing Research Overview.* ftp://download.intel.com/research/platform/terascale/terascale_overview_paper.pdf, 2006.

Chapter 6

Conclusion

High Performance Computing currently stands as a hot topic both for computer scientists and end users. The level of expectations is increasing, motivated by the noticeable technical advances and what is announced at the horizon. Harvesting a high fraction of the available processing power to solve real life problems is a central goal to achieve, as the gap between the theoretical performance and the sustained efficiency is more and more perceptible on modern supercomputers. From the scientific viewpoint, there are number of challenging achievements that are expected in order to come up with efficient and scalable computing solutions. Each involved topic is subject to intensive researches, with significant discoveries that are already effective. However, the connection among these individual advances need to be more investigated. This should be one of the major concern of future HPC investigations.

As we have so far demonstrated, solving large-scale problems in a short period of time using heterogeneous supercomputers is the main concern the high performance computing. We found that combining the advances in *continuous optimization* with suitable mathematical programming formulation of combinatorial problems remains the major approach in operation research. However, there is lack of studies on implementing state-of-the-art optimization methods on modern supercomputers. This is great technical challenge that I want to keep investigating. The branch-and-bound, for instance, is quite irregular and is likely to exhibit an elusive memory access pattern. Providing the right answer to the *load unbalanced* process, that will certainly show up from a standard scheduling, is a challenging task, but very important for *efficiency* and *scalability*. From a fundamental point of view, there is a need to reformulate problems accordingly, with a strong collaboration with people directly involved with real-life applications.

Another interesting topic we which to consider is *automatic code generation* for HPC. Programming current and future supercomputers is becoming more and more difficult, mainly because of their heterogeneity. In addition, obtaining a high fraction of the increasing peak performance is technically hard. One way to obtain an efficient code is to locally optimize each of its critical parts. Another way is to act at the code generation level. Tailoring a code to adapt or achieve the best possible performance on given architecture requires a complex set of program transformations, each designed to satisfy or optimize for one or more aspects (e.g. registers, cache, TLB, and instruction pipeline, data exchanges) of the target system. When the processing code is becoming complex, or when the target architecture is a combination of different processing units (hybrid or accelerated), it becomes very hard to handle the task by hand. Thus, it is highly expected to be able to achieve the necessary code transformations

in a systematic way. We plan to keep investigation this topic, which involves *compilation techniques*, *hardware comprehension*, and *performance prediction*.

We also plan to drive some research in *cloud computing*, *modeling and minimization of power consumption*, and *hierarchical memory profiling*. Study large-scale ill-conditioned matrix computation on supercomputers is another topic that needs to be addressed.

Personnal bibliography

6.1 Regular papers

1. Claude Tadonki, and Bernard Philippe, *Parallel multiplication of a vector by a Kronecker product of matrices*, Parallel Distributed Computing Practices PDCP, volume 2(4), 1999.

2. Sanjay Rajopadhye, Tanguy Risset, et Claude Tadonki, *Algebraic Path Problem on linear arrays*, Techniques et Sciences Informatiques TSI, 20 (5), 2001.

3. Claude Tadonki, and Bernard Philippe, *Parallel multiplication of a vector by a Kronecker product of matrices (part II)*, Parallel Distributed Computing Practices PDCP, volume 3(3), 2000.

4. C. Tadonki, *A Recursive Method for Graph Scheduling*, Scientific Annals of Cuza University, Vol 11, p.121-131, 2002.

5. C. Beltran, C. Tadonki and J.-Ph. Vial, *Solving the p-median problem with a semi-Lagrangian relaxation*, Computational Optimization and Applications, Volume 35(2), October 2006.

6. F. Babonneau, C. Beltran, A. Haurie, C. Tadonki and J.-P. Vial, *Proximal-ACCPM: a versatile oracle based optimization method*, Computational and Management Science, Volume 9, 2007.

7. C. Tadonki, *Mathematical and Computational Engineering in X-Ray Crystallography*, International Journal of Advanced Computer Engineering, volume 1(2) 2008.

8. T. Saidani, L. Lacassagne, J. Falcou, C. Tadonki, S. Bouaziz, *Parallelization Schemes for Memory Optimization on the Cell Processor : A Case Study on the Harris Corner Detector*, Transactions on High-Performance Embedded Architectures and Compilers, volume 3(3) 2008.

9. C. Tadonki, *Integer Programming Heuristic for the Dual Power Setting Problem in Wireless Sensors Networks*, International Journal of Advanced Research in Computer Engineering, vol 3(1) 2009.

10. E. Dadi, M. Daoudi, C. Tadonki *3D Shape Retrieval using Bag-of-feature method basing on local codebooks*, International Journal of Future Generation Communication and Networking, Vol. 5(4), December, 2012.

6.2 International conference papers

1. Sanjay Rajopadhye, Tanguy Risset, et Claude Tadonki, *The algebraic path problem revisited*, European Conference on Parallel Computing Europar99, Toulouse (France), Lncs Sringer-Verlag, N 1685, p. 698-707, August 1999.

2. Claude Tadonki, *Parallel Cholesky Factorization*, Parallel Matrix Algorithms and Appliations PMAA Worshop, Neuchatel (Switzerland), August 2000.

3. Claude Tadonki, et Bernard Philippe, *Méthodologie de conception d'algorithmes efficaces pour le produit tensoriel*, CARI2000, Tananarive (Madagascar), Octobre 2000.

4. Patrice Quinton, Claude Tadonki, et Maurice Tchuente, *Un échéancier systolique et son utilisation dans l'ATM*, CARI2000, Tananarive (Madagascar), Octobre 2000.

5. Claude Tadonki, *A Recursive Method for Graph Scheduling*, International Symposium on Parallel and Distributed Computing (SPDC), Iasi, Romania, July 2002.

6. L. Drouet, A. Dubois, A. Haurie and C. Tadonki, *A MARKAL-Lite Model for Sustainable Urban Transportation*, Optimization days, Montreal, Canada, May, 2003.

7. Claude Tadonki, *ProxAccpm: A convex optimization solver*, International Symposium on Mathematical Programing, ISMP2003, Copengagen, Danmark, August 2003.

8. C. Tadonki, M. Singh, J. Rolim and V. K. Prasanna, *Combinatorial Techniques for Memory Power State Scheduling in Energy Constrained Systems*, Workshop on Approximation and Online Algorithms (WAOA), Budapest, Hungary, September 2003.

9. R. Ndoundam, C. Tadonki, and M. Tchuente, *Parallel chip firing game associated with n-cube orientation*, International Conference on Computational Science, ICCS04 (LNCS/Springer), Krakow, Poland, June 2004 .

10. O. Briant, C. Lemarchal,K. Monneris,N. Perrot,C. Tadonki,F. Vanderbeck,J.-P. Vial,C. Beltran,P. Meurdesoif, *Comparison of various approaches for column generation*, Eigth Aussois Workshop on Combinatorial Optimization, 5-9 january 2004.

11. C. Tadonki and J.-P. Vial, *Efficient algorithm for linear pattern separation*, International Conference on Computational Science, ICCS04 (LNCS/Springer), Krakow, Poland, June 2004.

12. C. Beltran, C. Tadonki, J.-P. Vial, *Semi-Lagrangian relaxation*, Workshop on Computational Econometrics and Statistics, Neuchatel, Switzerland, April 2004.

13. C. Tadonki, C. Beltran and J.-P. Vial , *Portfolio management with integrality constraints*, Workshop on Computational Econometrics and Statistics, Link, Neuchatel, Switzerland, April 2004.

14. C. Beltran, C. Tadonki and J.-Ph. Vial, *The p-median problem solved by semi-Lagrangian relaxation*, First Mathematical Programming Society Int. Conference on Continuous Optimization (ICCOPT I), Troy, USA, August 2-4, 2004.

15. C. Tadonki and J. Rolim, *An analytical model for energy minimization*, III Workshop on Efficient and Experimental Algorithms (WEA04), Angra dos Reis, Rio de Janeiro, Brazil, May 2004.

16. C. Tadonki , *Universal Report: A Generic Reverse Engineering Tool*, 12th IEEE International Workshop on Program Comprehension (IWPC), Bari, Italy , June 2004.

17. C. Tadonki and J. Rolim, *An integer programming heuristic for the dual power management problem in wireless sensor networks*, 2nd Int. Workshop on Managing Ubiquitous Communications and Services, MUCS2004, Dublin, Ireland, December 13, 2004.

18. C. Tadonki, *Off-line settings in wireless networks*, 3rd Int. Symposium on Computational Intelligence and Intelligent Informatics, ISCIII2007, Agadir, Morocco, March 28-30, 2007.

19. T. Saidani, J. Falcou, C. Tadonki, L. Lacassagne, and D. Etiemble, *Algorithmic Skeletons within an Embedded Domain Specific Language for the CELL Processor*, Parallel Architectures and Compilation Techniques (PACT), PACT09, Raleigh, North Carolina (USA), September 12-16, 2009.

20. C. Tadonki, G. Grosdidier, and O. Pene, *An efficient CELL library for Lattice Quantum Chromodynamics*, International Workshop on Highly Efficient Accelerators and Reconfigurable Technologies (HEART) in conjunction with the 24th ACM International Conference on Supercomputing (ICS), pp. 67-71, Epochal Tsukuba, Tsukuba, Japan, June 1-4, 2010.

21. C. Tadonki, L. Lacassagne T. Saidani, J. Falcou, K. Hamidouche, *The Harris algorithm revisited on the CELL processor*, International Workshop on Highly Efficient Accelerators and Reconfigurable Technologies (HEART) in conjunction with the 24th ACM International Conference on Supercomputing (ICS), pp. 97-100, Epochal Tsukuba, Tsukuba, Japan, June 1-4, 2010.

22. C. Tadonki, *Ring pipelined algorithm for the algebraic path problem on the CELL Broadband Engine*, Workshop on Applications for Multi and Many Core Architectures (WAMMCA 2010) in conjunction with the International Symposium on Computer Architecture and High Performance Computing , Petropolis, Rio de Janeiro, Brazil, October 27-30, 2010.

23. C. Tadonki, *Large Scale Kronecker Product on Supercomputers*, 2nd Workshop on Architecture and Multi-Core Applications (WAMCA 2011) in conjunction with the International Symposium on Computer Architecture and High Performance Computing (SBAC PAD 2011), Vitria, Esprito Santo, Brazil, October 26-29, 2011.

24. D. Barthou, G. Grosdidier, M. Kruse, O. Pene and C. Tadonki, *QIRAL: A High Level Language for Lattice QCD Code Generation*, Programming Language Approaches to Concurrency and Communication-cEntric Software (PLACES'12) in conjunction with the European joint Conference on Theory & Practice of Software (ETAPS), Tallinn, Estonia, March 24-April 1, 2012.

25. C. Tadonki, *Basic parallel and distributed computing curriculum*, Second NSF/TCPP Workshop on Parallel and Distributed Computing Education (EduPar'12) in conjunction with the 26th IEEE International Parallel & Distributed Processing Symposium (IPDPS), Shanghai, China, May 21-25, 2012.

26. C. Tadonki, L. Lacassagne, E. Dadi, M. Daoudi *Accelerator-based implementation of the Harris algorithm*, 5th International Conference on Image Processing (ICISP 2012), Agadir, Maroc, June 28-30, 2012.

27. E. Dadi, M. Daoudi, C. Tadonki *3D Shape Retrieval using Bag-of-feature method basing on local codebooks*, 5th International Conference on Image Processing (ICISP 2012), Agadir, Maroc, June 28-30, 2012.

6.3 National conference papers

1. Claude Tadonki, *Systme d'quations rcurrentes et multiplication parallle d'un vecteur par un produit tensoriel de matrices*, Rencontres Francophones de Parallelisme Renpar'11, Rennes (France), 1999.

2. Claude Tadonki, *Ordonnancements canoniques*, Renpar12, Rencontres Francophones de Parallelisme, Besanon (France), Juin 2000.

3. Claude Tadonki, *Complexit des ordonnancements canoniques et drivation d'architecture*, Rencontres Francophones de Parallelisme Renpar13, Paris (France), Avril 2001.

6.4 Book or Chapter

1. Ordonnancement pour l'informatique parallèle, Chap: *Une approche récursive de synthèse d'ordonnancements parallèles* Editeur HERMES SCIENCE PUBLICATIONS / LAVOISIER (2-7462-0730-3) Date de parution: 10-2003 202p. 16x24 Reli´.

6.5 Posters and communications

1. C. Tadonki, *Refinement experiments with RADDAM data*, EMBL bilateral meeting, Hamburg, Germany, June 26-28, 2006.

2. C. Tadonki, G. Grosdidier, and O. Pene, *Large scale Lattice Quantum Chromodynamics*, the 24th ACM International Conference on Supercomputing (ICS), pp. 67-71, Epochal Tsukuba, Tsukuba, Japan, June 1-4, 2010.

3. C. Tadonki, *Automatic LQCD Code Generation*, Quatrimes Rencontres de la Communaut Française de Compilation

6.6 Research reports

1. C. Tadonki, and B. Philippe, *Parallel multiplication of a vector by a Kronecker product of matrices*, IRISA report n 1194, 1998.

2. P. Quinton, C. Tadonki, et M. Tchuente, *Un échéancier systolique et son utilisation dans l'ATM*, IRISA report n 1348, 2000.

3. C. Tadonki, *Synthse d'ordonnancements parallles par reproduction canonique*, IRISA report n 1349, also INRIA report n 3996, 2000.

4. D. Cachera, S. Rajopadhye, T. Risset, and C. Tadonki, *Parallelization of the algebraic path problem on linear simd/spmd arrays*, IRISA report n 1409, 2001.

5. C. Tadonki and J.-P. Vial, *The linear separation problem revisited with accpm*, Cahier de Recherche n 2002.11, University of Geneva, June 2002.

6. F. Babonneau, C. Beltran, O. du Merle, C. Tadonki and J.-P. Vial, *The proximal analytic center cutting plane method*, Technical report, Logilab, HEC, University of Geneva, 2003.

7. C. Beltran, C. Tadonki, and J.-P. Vial, *Semi-Lagrangian relaxation*, Technical report, Logilab, HEC, University of Geneva, 2004.

List of Figures

List of Tables